Knowledge Discovery from Sequential Data

Frank Höppner

Vom Fachbereich für Mathematik und Informatik der Technischen Universität Braunschweig genehmigte Dissertation zur Erlangung des Grades eines Doktor-Ingenieurs.

1. Referent:	Prof. Dr. F. Klawonn
2. Referent:	Prof. Dr. R. Kruse
Eingereicht am:	1. Juli 2002

17. Januar 2003

Contents

Abstract: A new framework for analyzing sequential or temporal data such as time series is proposed. It differs from other approaches by the special emphasis on the interpretability of the results, since interpretability is of vital importance for *knowledge* discovery, that is, the development of new knowledge (in the head of a human) from a list of discovered patterns. While traditional approaches try to model and predict *all* time series observations, the focus in this work is on modelling local dependencies in multivariate time series. This makes it possible to deal with irregular or chaotic series. The proposed discovery process consists of (1) time series abstraction to get a representation close to the human perception of time series, (2) the enumeration and ranking of qualitative relationships in the data, (3) the specialization with quantitative constraints and the generalization of patterns to overcome limitations that are implicitly induced by the search bias.

Zusammenfassung: In dieser Arbeit wird ein Ansatz zur Analyse sequentieller oder zeitlicher Daten (etwa Zeitreihen) vorgestellt. Er unterscheidet sich von anderen Ansätzen in der besonderen Berücksichtigung der Interpretierbarkeit der Ergebnisse, weil dies für die *Wissen*entdeckung, also die Entwicklung neuen Wissens (im Kopf eines Menschen) aus einer Liste von entdeckten Mustern, entscheidend ist. Während traditionelle Ansätze versuchen, ein globales zugrundeliegendes Modell für die *gesamte* Zeitreihe zu finden, liegt der Schwerpunkt hier auf der Modellierung lokaler Zusammenhänge. Dadurch können auch irreguläre oder chaotische Systeme untersucht werden. Der Prozeß besteht aus (1) der Abstraktion der Zeitreihen, um der Art der Wahrnehmung duch den Menschen besser gerecht zu werden, (2) der Aufzählung und Bewertung qualitativer Zusammenhänge in den Daten, (3) der Spezialisierung mit quantitativen Nebenbedingungen und der Generalisierung von Mustern zur Überwindung von Einschränkungen, die implizit durch die Definition des Suchraumes gegeben sind.

Symbols

The meaning of frequently used symbols or notations is given in the table below. If nothing else is stated in the context, the symbol in the first column has the meaning given in the third column. The second column denotes the page number (if any) where the symbols are explained and defined formally.

Symbol	Page	Meaning
$\mathbb{R}$		the set of real numbers
$\mathcal{I}$	46	set of interval relationships
$\mathcal{I}_C$	74	$\mathcal{I}\backslash\{\textit{after}, \textit{before}, \textit{contains}, \textit{during}\}$
$\mathcal{I}_L$	101	$\mathcal{I}\backslash\{\textit{after}, \textit{before}\}$
ir	46	Given two intervals I_1 and I_2, $\text{ir}(I_1, I_2) \in \mathcal{I}$ denotes their interval relationship.
W	50	content of sliding window (temporal pattern)
$P \sqsubseteq W$	48	pattern P is subpattern of sliding window W
Δt_{win}	50	sliding window width
t_{act}	51	current value of reference point (right bound) of sliding window
F_k	61	set of frequent patterns of size k
C_k	61	set of candidate patterns of size k
$\mathcal{S}$	45	set of interval labels
O_P	52	support set of pattern P
L		number of intervals in the interval sequence
$y \leftarrow\!\shortmid x$		assign expression x to variable y
$\rightarrow x$		x is an input variable of a procedure or function (call by value)
$\leftarrow x$		x is an output variable of a procedure or function (return value)
$\leftrightarrow x$		x is an input and output variable of a procedure or function (call by reference)

Chapter 1

Introduction

Knowledge discovery is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [Fayyad et al., 1996]. In this work we consider sequential (or temporal) data such as time series and are particularly concerned about the question how to construct a computer program that supports a human in discovering interdependencies in multivariate time series. Our focus will be on measurements over a long period of time that are obtained from continuously observing the environment or technical systems. Computer programs can perform numeric and symbolic transformations quickly, but new knowledge arises in the head of the human only. Therefore, special emphasis lies on the interpretability of the patterns to be identified in the data.

To forecast a systems behaviour in the future it is probably the best to develop a model of the system and to estimate its parameters from observations in the past. While in some simple cases (unmeasured) environmental influence may be that small that it is indeed possible to develop such a model and to predict the value of a variable B from the development of variables A_i in the past, in arbitrary knowledge discovery applications we should neither expect simplicity nor completeness of information, since often unpredictable circumstances greatly influence the value of B, rather than the few signals A_i that we were able to measure and record cheaply. Quite often the data has not even been collected for the purposes of further analysis. Thus, accepting that there are more things influencing B that we may be aware of, what is left for us to do? At least we know that systems (or subsystems) often cycle through a number of internal states that lead to repetitions of certain patterns in the observed variables. Observing or discovering these patterns may help a human to resolve the underlying causal relationships (if there are any). Rather than trying to explain the behaviour of the variables *globally*, we therefore seek for *local dependencies* or *local patterns* that can be observed frequently[1]. Having found such dependencies, an expert in the field may examine what has been found and judge about its importance or relevance. Since correlations do not necessarily point out cause-effect relationships, the final judgement by an expert is very important. On the other hand, an expert in the field is not necessarily familiar with whatever kind of analysis we are going to use, therefore it is important to obtain results that are easily understandable for the expert.

The visual pattern recognition capabilities of the human brain are truly remarkable and still outperform most artificial approaches to pattern recognition available up

[1]The term *frequently* should not be taken too seriously, we refer to patterns that occur in a certain percentage p of all observations, but p can be as small as 1%, for example.

to now. It is therefore quite natural for a human to turn temporal information into a graphical representation (e.g. time series profile) and to apply this impressive ability to the analysis of time-varying data.[2] When listening to human experts arguing about time series they frequently use terms like "linearly increasing segment" or "exponentially decreasing segment" to describe the graph. Thus, experts seem to be used in arguing on the *shape* or *appearance* of time series profiles, since changes in the system manifest in local trends in the observed variables. To support an expert in the analysis of temporal data, we will use a similar shape-based notion of time series patterns.

Many people are familiar with the situation of having not enough knowledge to predict the system in general, but being aware of certain qualitative relationships between the measured variables on the one hand and the systems behaviour in the future on the other. May be they have already some snapshots of typical behaviour in mind and seek for certain patterns in the data as indications for possible future events. Such *typical key situations* are often used by humans to control technical systems simply by visual inspection of displayed trends [Bakshi and Stephanopoulos, 1995]. Even if considerable knowledge about the underlying process is available, measuring and processing all the necessary inputs may be difficult and expensive. When using only those inputs that we can measure cheaply, we know that we cannot provide a good global model but it might still be possible to get a good *local model* for particular situations – not explaining everything but at least more than without any model. An example for this case is local weather forecasting. We get predictions of the next days weather on TV every day, but considerable computational power is necessary to run the large simulations. And still, it is not possible to make a precise prediction if the storm will hit me at what time and what strength right here where I am sailing. Experienced sailors have therefore developed prediction rules that base on the overall outlook, the season, local weather reports, etc., and the *shape of the air pressure curve*. Measuring the overall outlook is subjective and automatization is not straightforward, analyzing the graphical picture of the weather situation (e.g. cloudiness, white crests, dust, etc.) by a program is technically intricate. But electronic barographs are cheap. And *if* it is possible to predict only some of the dangerous situations with acceptable reliability by using only a subset of (cheaply measured) variables, an unexperienced sailor would be glad to have a device that throws an alarm in such cases.

Detecting similar patterns in long multivariate data sequences (this includes numerical time series as well as symbolic sequences) seems to be a promising way to induce some knowledge about the system under consideration. However, acquiring such knowledge manually takes a lot of time and overburdens a human very soon, even if the data volume is just moderate. But although the detailed inspection of large data sets is often skipped, a human still performs well in developing hypotheses, verifying them, drawing conclusions, designing experiments, etc. – that is, working on the basis of a shape-based representation in general. The aim is thus to support a human in the weakest link of the analysis chain, namely the treatment of large data volumes, without forcing to adopt to new representations of the data but allowing to keep the representation a human is used to. In this way, we hope to maximize the benefits of human expertise in visual pattern recognition and processing. The human can be supported by pointing out interesting local patterns and rules about the qualitative/semi-quantitative behaviour of the series. These are very tedious tasks humans usually do not like, since ranking of different hypotheses (temporal patterns or rules) requires carefully scanning through all the data at

[2] We exclude highly periodic signals from our considerations, where a frequency spectrum is usually more informative than the time profile itself.

hand.

The challenging aspects of this task include at least the following subtasks: (i) the generation of an appropriate data abstraction (in case of time series) that corresponds to the perception of a human, (ii) the definition of the pattern space whose elements we want to find the abstracted data, (iii) the efficient mining for patterns in the data, and (iv) the identification of potentially interesting rules about local interdependencies. These steps, amongst others, will be discussed in the subsequent chapters.

1.1 Motivation: Why addressing local shape?

The problem of finding common characteristics of multiple time series or different parts of the same series requires a notion of similarity. Two time series can be considered similar if their values at each point in time do not differ by more than a small $\varepsilon \in \mathbb{R}_{>0}$. This is a reasonable definition of time series similarity, useful in many applications, e.g. [Keogh et al., 2000, Joentgen et al., 1999, Faloutsos et al., 1994]. Besides point-to-point similarity we can find two profiles to be similar due to their *structural similarity*. If we focus on the shape, two sine waves are more similar to each other than a sine wave and a linear function. It is this kind of similarity that we are addressing in this work, because we think that point-to-point similarity is not sufficient for many systems with complex dynamics, as we will explain in this section.

It has been observed by many authors that humans are very good in identifying structural or qualitative similarity of time series. While the plain record of the quantitative values over time does not invoke appreciable levels of cognitive activity to a human, by visual inspection of displayed trends a human operator is capable of controlling processes [Bakshi and Stephanopoulos, 1995]. There is psychological evidence in the literature [Attneave, 1954] that humans decompose time series into a sequence of simple basic profiles (e.g. straight lines), such that all points in each segment behave similar or follow the same local trend. Each of these few segments is usually simple in shape and easy to grasp. Many examples for applications where shapes, qualitative, structural or abstracted descriptions of time series are used to reason about the underlying process can be found in the literature:

- Skippers use rules that consider the qualitative behaviour of the airpressure curve for short-term local weather forecast [Karnetzki, 1999].

- Human operators perceive fermentation processes as a set of shapes of a few key variables and reason about the process on qualitative interpretation of these shapes [Bakshi and Stephanopoulos, 1995, Konstantinov and Yoshida, 1992].

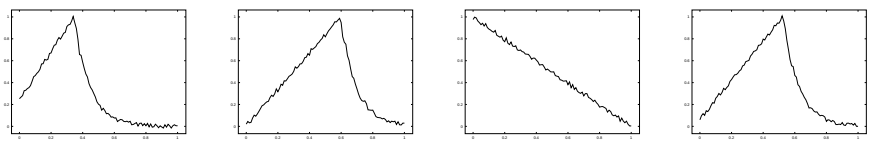
- The theory of qualitative reasoning has shown that a large part of expert knowledge consists of qualitative descriptions of continuous variables [Kuipers, 1994, Falkenhainer and Forbus, 1991, Forbus, 1987, Forbus, 1981]. (Rather than *using* qualitative knowledge to explain what is happening or what may happen, here we are interested in *inducing* such knowledge from data.)

- In medical diagnosis and health care a large quantity of clinical information about patients is routinely collected, such as multivariate ECG data.

Temporal patterns are used by the physicians to distinguish various diseases or to gain decision support for medication [Guimarães and Ultsch, 1999, Shahar and Musen, 1996, Bellazzi et al., 2002, Carrault et al., 2002].

- In material science [Capelo et al., 1998] qualitative features of experimental data are used to select an appropriate physical model for the material out of a set of plausible candidate models.
- To access subterranean oil trapped in some geological formations, well tests (pressure transient tests) are performed to evaluate the state of the surrounding ground formation. Pressure-time profiles are analysed visually to interpret the results [McIlraith, 1989].
- In the technical analysis of stock data typical patterns are characterized by a few critical points, that are perceptually important in the human identification process. A list of typical pattern is used to match them against stock data and predict future events [Chung et al., 2001, Povinelli, 2000].
- In mission operations for NASA's Space Shuttle many thousand sensors are telemetered once per second. The taped data is considered useful for diagnosis or anomaly detection. In [Keogh and Smyth, 1997] an approach to time series retrieval is proposed, where the queries consist of piecewise linear shapes.

But it is not only true that humans are used to a shape-oriented perception of time series, but this approach seems to be beneficial from a technical or practical point of view, also. As an example, consider a technical systems that is run several times. Usually such systems are not fully deterministic in their behaviour. Even if we can observe at some points in the system Gaussian distributed measurements (e.g. to which degree a valve has been opened), this does not necessarily manifest in a Gaussian distribution of the output value at a fixed time. The degree of opening controls the amount of water that flows through a pipe and thereby influences the point in time when the tub is full, which then may cause some other action. Many variables are therefore subject to translation or dilatation in time. Those measures used traditionally for estimating similarity (e.g. pointwise Euclidean norm) will fail in providing useful hints about the similarity of the underlying processes. Figure 1.1 shows an example motivated by [Keogh and Pazzani, 1999b]. Clustering of the profiles by a human would lead to the partition $\{\{a, b, d\}, \{c\}\}$, but when using the Euclidean norm to $\{\{a, c\}, \{b, d\}\}$ – the varying slope of the linear increase and the changing position of the maximum distorts the Euclidean measure. With respect to the remarks we made before, it seems much more likely that the profiles a, b, d characterize similar situations of the process than profiles a and c.



(a) (b) (c) (d)

Figure 1.1: Similarity of time series: in terms of Euclidean distance, (a) is closer to (c) than to (b) or (d).

The main reasons for using structural or shape-based information can be summarized as follows:

- Shapes are more robust against dilatation effects.
- Shapes are more robust against noise, since they are usually obtained from a set of measurements rather than from one or two values.
- Quite often there is only a limited number of sensors, most variables of the process remain unobserved or are extremely difficult to incorporate into an automatic reasoning process. It is therefore important to get most out of the given measurements, that is, instead of considering global properties or single values we also use the local behaviour of the profile.
- Extracting structural features from the time series allows for a natural language description and thus a direct interpretation by the expert (comprehensibility).

And why *local* shapes? Very long shape patterns are more difficult to remember and understand by a human. More important, in many cases it is believed that events which happened considerable time ago do no longer affect the current state of the system. Similar assumptions can be found in the theory of Markov chains or Dynamic Bayesian networks, for instance. The theory of chaos states that, while it is impossible to make conclusions far in the future in a chaotic system, it is still possible to make short-term predictions. For instance, the atmospheric air pressure system belongs to the chaotic processes, which is the reason why long-term forecasts usually fail. Local rules, on the other hand, are useful and reliable [Karnetzki, 1999].

1.2 Problem Statement: What are we exactly looking for?

We seek for local descriptive summaries (local models or patterns) in a long (multivariate or univariate) *sequence of observations*, such as *interesting* rules

$$\text{if } P \text{ then } Q \text{ with probability } p$$

to locally *predict* a temporal pattern Q in the conclusion from an occurrence of a temporal pattern P in the premise. The temporal patterns P and Q consist of qualitative and/or quantitative attributes that hold in the original sequence over a certain time interval.

We want to explain some points of this problem statement in greater detail.

- *Temporal patterns* consists of shape descriptions that hold over a specific time interval and their temporal relationship to each other. For instance, if the premise pattern is "constant, non-zero waterflow", the conclusion pattern may be "bath tub water-level increases linearly while constant, non-zero waterflow is observed".
- The term *prediction* includes not only forecasting of the future (but this will be our main focus). If we are interested in detecting a certain event B that we cannot measure directly, it may manifest on other (measurable) attributes A_i that appear *after* the occurrence of B. In such a case, a rule may be used to "predict" an event in the past. To distinguish both cases formally, we require that the temporal pattern in the conclusion comprises the temporal relationship of "conclusion attributes" to those in the premise.

- A human will finally judge about the *interestingness* of a rule, however, if a the number of rules is large, which should we analyze first? Thus, by "interestingness" we refer to an objectively measurable number that may address our belief in the rule (the rule probability p) or aspects like "How often can we apply the rule?" and can be used to rank the rules.

- A *sequence of observations* is not necessarily measured at a constant sampling rate. Constant sampling rates are quite usual in technical systems, but in health care, for instance, the height of a child is measured frequently when the child is young, and much less frequent when the child becomes older [Ramsay and Silverman, 1997]. After extraction of temporal attributes that hold over a time interval it does not matter if it has been extracted from a uniformly or irregularly sampled series. In this generality a sequence of observation may – besides classical time series – also contain rare but important events that may have influence on the variables.

- We have motivated the focus on shapes (that is, qualitative attributes) in section 1.1, however, in case of discrete-valued variables there are no "shapes". We do not exclude discrete-valued data from our considerations, since such a variable also holds a specific value (rather than a shape) over a time interval. Besides that, any attribute (real- or discrete-valued) may carry additional (quantitative) information about the time interval that is covered (e.g. slope of a linear segment, dosage of an admixture in a chemical process, etc.), which might play a role for the discovery of interesting rules.

1.3 Related Work: Where do we have similar problems?

Our objective is to induce relationships between patterns occurring in temporal or sequential data. Shapes in time series profiles persist over time intervals, thus our initial data can be characterized as a sequence of labeled intervals: The labels address the kind of pattern or shape that can be observed in the respective time interval. Most of the (rule) induction methods in machine learning, however, assume static data, that is, they do not consider time explicitly. Compared to static data and even to temporal/sequential data, little work has been done to analyse interval data.

In traditional time series analysis the goal of function approximation (regression) or analytical models is to predict the next value of the time series. (Regression methods will be discussed in section 2.2.1.) Analytical models like the l^{th} moving average (MA) model assume that the predicted output is a linear combination of the last l systems outputs. Autoregressive (AR) models use the last l predicted outputs and the current system output to predict the next output. Combinations are also possible (ARMA, ARIMA models [Chatfield, 1989]). While linear, time-invariant systems can be described by such models, the models cannot get around with non-stationarity and nonlinearity (or multivariate time series). Since we do not assume stationary time series, nor try to explain the underlying process as a whole but seek for *local* and *descriptive* dependencies in multivariate time series, these methods are not suited for our purposes.

There is a variety of approaches that deal with rule discovery in temporal data such as time series. Standard techniques for rule induction (e.g. decision trees) have been adopted for temporal data in [Kadous, 1999, Karimi and Hamilton, 2000,

Savnik et al., 2000]. To characterize a variable's value $x(t)$ over time, consecutive measurements of x are embedded in a vector, e.g. $(x(t-2\Delta), x(t-\Delta), x(t))$. The static rule induction algorithms (like ID3 [Quinlan, 1986], C4.5 [Quinlan, 1993], CN2 [Clark and Niblett, 1989], AQ11, etc.) can then be used to learn rules that reflect temporal dependencies (see e.g. [Kadous, 1999, Karimi and Hamilton, 2000, Savnik et al., 2000]). A certain attribute used in an induced rule may then address the value of x five minutes ago, for instance. However, temporal processes are often subject to dilatation in time, that is, in similar situations events may occur a bit earlier or later than in the past. If we simply check for a value at a specific point in time, there is no way to cope with such translation and dilatation effects. Similar problems occur when using classical distance measures for time series similarity, like Euclidean distance or (auto)correlation [Martinelli, 1998]. A classic approach to handle such effects is dynamic time warping [Sankoff and Kruskal, 1983, Berndt and Clifford, 1996]. For the retrieval of similar time series, novel distance measures have been proposed (e.g. [Agrawal et al., 1993, Agrawal et al., 1995, Keogh and Pazzani, 1999a, Kim et al., 2000, Keogh et al., 2000]), but the additional effort of discovering knowledge (compared to the simpler task of retrieving similar series) forces most approaches to rely on more simple measures like a (slightly modified) Euclidean distance (e.g. [Das et al., 1998]).

Since we search for all potentially interesting rules, we have to run through the large hypothesis space of possible rules efficiently (which is the so-called data mining step). We will adopt techniques from the discovery of association rules [Agrawal et al., 1996] to the problem of finding rules on temporal patterns in interval sequences. Association rules are rules of the kind "most of the people that buy butter and milk, also buy bread", that is, associations between sets of items (purchased products in this example) are described. Association rules have been generalized to work with "time stamped data", that is data ordered in time [Mannila et al., 1997, Srikant and Agrawal, 1996]. The work of Mannila et. al is probably most closely related to our work of temporal pattern discovery. They consider sequences of events (like alarms in telecommunication networks) and seek for temporal event dependencies. Even for event sequences interval representations may be of interest as the number of events increases: In [Mannila and Salmenkivi, 2001] the events are examined with respect to intervals of equal occurrence density. Then, either the change points in density can be considered as a derived event sequence or the intervals themselves can be used for further analysis.

A number of extensions for event sequence mining has been proposed, for instance in [Li et al., 2000] periodic temporal patterns in hierarchical calendar time are considered and in [Rainsford and Roddick, 1999] the temporal distribution of transactions is summarized by means of temporal logic. Recently some work has been published about analysing interval data. All approaches use Allen's interval logic or a subset or variation thereof. In [Cohen, 2001] temporal patterns are identified via a statistical test. In [Guimarães and Ultsch, 1999] the time series segmentation is learned via neural networks before grammatical rules are inferred. In [Kam and Fu, 2000] a (limited) set of interval patterns is learned from a database of short interval sequences. In [Villafane et al., 2000, Villafane et al., 1999] only a containment-relationship is examined. In our approach, we generalize the discovery of episodes in event sequences [Mannila et al., 1997] to interval sequences [Höppner, 2001a], where we consider a much more flexible definition of temporal pattern compared to [Kam and Fu, 2000].

As already mentioned, a closely related area is that of approximate matching of (short) queries to (long) time series. Due to noise and varying environmental conditions the same experiment may look different each time when compar-

ing measurements over time. There has been a lot of work on subsequence matching [Agrawal et al., 1993, Agrawal et al., 1995, Berndt and Clifford, 1996, Shatkay, 1995, Faloutsos et al., 1994, Keogh and Pazzani, 1999b, Keogh, 1997]. However, subsequence matching requires an initial pattern to be given by an expert and the problem is restricted to find all subsequences that deviate from this initial pattern by no more than a certain threshold. These patterns can be used to verify a temporal rule an expert had in mind. However, verification has the limitation that we have to guess the rule before we can verify it, thus we can miss interesting rules. All the cited papers deal with continuous features, the time series itself or a transformation, its Fourier transform for instance.

In this work we will concentrate on structural or qualitative descriptions of a time series. In the literature, we have found the following kind of abstractions:

- Use minima, maxima, inflexion points, etc. for segmentation. The basic patters are then defined a priori [Bakshi and Stephanopoulos, 1995, McIlraith, 1989, Capelo et al., 1998].
- Descriptions can be learned from a set of examples (labeled attribute vector or labeled subsequence) [Guimarães and Ultsch, 1999]. A new time series is pushed through the learned classifier and the subdivision of the profile is made whenever the algorithm votes for a different class.
- In absence of any prior knowledge patterns can be found automatically by means of clustering short subsequences [Das et al., 1998], which is some kind of "subsequence quantization". Neighbouring subsequences that belong to the same cluster give us the segmentation.

Other related work, although not relevant for our purposes, include the mining of interval data (mining traditional transaction data where some of the variables have interval attributes) [Miller and Yang, 1997].

1.4 Major Contributions and Outline of Process

As already mentioned, knowledge discovery is understood as a process that is usually iterated several times before new useful knowledge has been induced from data [Fayyad et al., 1996]. Figure 1.2 depicts a graphical outline of the process proposed in this thesis. We assume that data cleaning and preprocessing has already been done (see e.g. [Pyle, 1999]), including the removal of outliers but not necessarily noise. Our initial data (top left of figure 1.2) may consist of time series (not necessarily uniformly sampled) as well as singular events.

One cycle through the process consists of the selection of a set of shapes or primitive patterns that seems to be useful to argue about the series at hand (either learned from data or defined a priori) and the abstraction of time series according to this set. Then, all temporal patterns in the stream of labeled intervals are sought (association mining step) that occur more often than a certain threshold. From these so-called frequent patterns (forecasting-) rules can either be derived directly or after additional specialization or generalization steps. Usually, one gets a large number of patterns and rules and has to put some effort in ranking them to pick only the best rules among them. After that, the discovered patterns are presented to a human expert of the field, who inspects the rules, develops new hypotheses or ideas, and the process is restarted with a changed set of labels or features that are extracted from the time series.

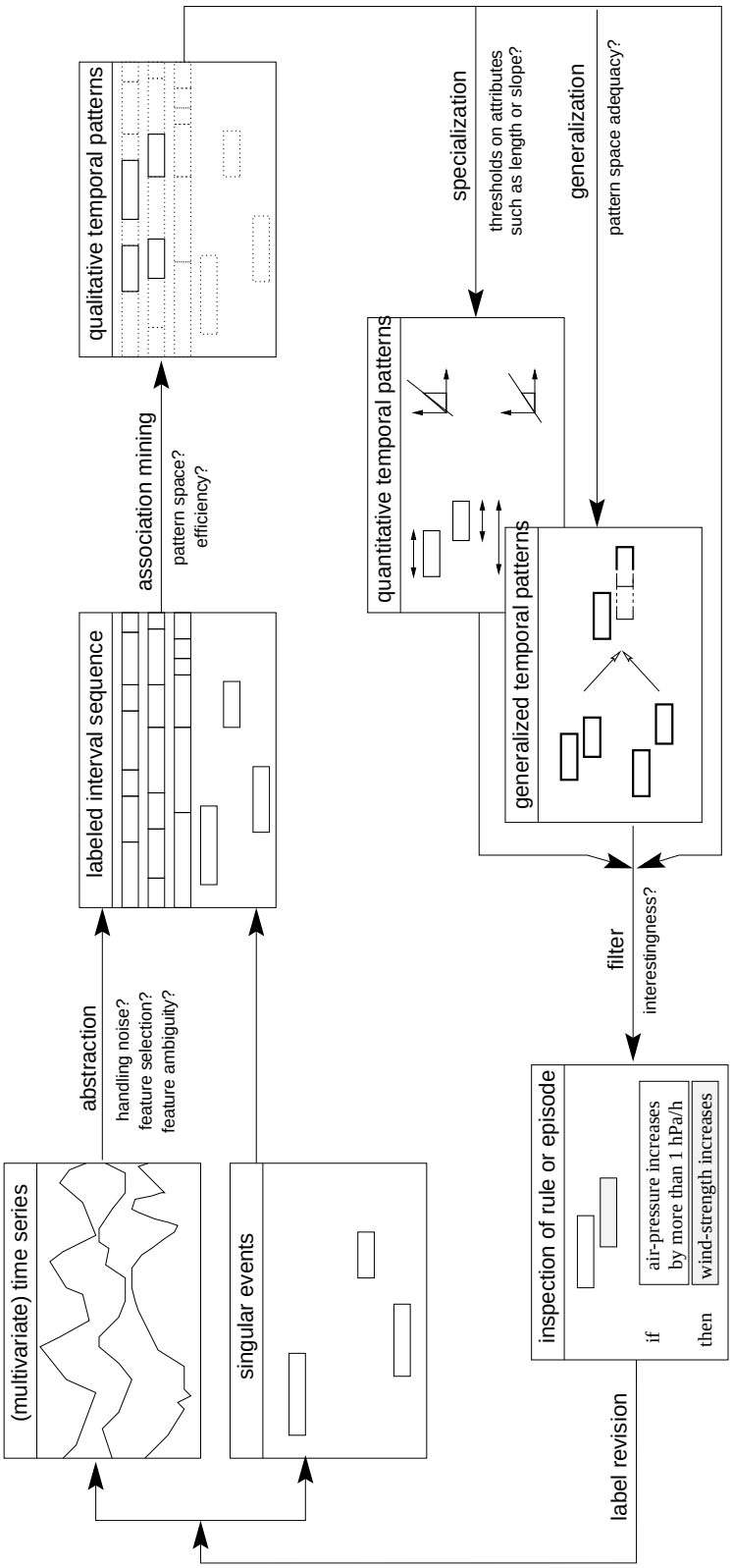


Figure 1.2: Outline of the Knowledge Discovery Process.

The major contributions of this work to the field and differences to other work are

- **Consequent use of abstractions (interval sequences)**: Interval sequences, although useful for abstracting time series and supporting a higher level of understanding, are seldomly used in knowledge discovery from time series. They are sometimes used in the retrieval of similar time series [Keogh and Pazzani, 1999b, Keogh et al., 2000], and more often in the medical domain [Kam and Fu, 2000, Guimarães and Ultsch, 1999, Bellazzi et al., 2002]. Especially in the medical domain expert knowledge is quite often necessary in early stages to fix thresholds and make design decisions (which limits the usefulness of these approaches for arbitrary knowledge discovery tasks). In many cases the data is transformed into representations that suit the respective (static) data mining algorithms, that is, the data is transformed such that standard algorithms can be used that were not designed to work with sequential data (e.g. [Bellazzi et al., 2002, Rainsford and Roddick, 1999, Kadous, 1999]). In the worst case the discovered patterns are then difficult to understand resp. verify by a human, who would not have formulated rules of this kind by himself. In this work we develop new algorithm explictly tailored to the needs of temporal data analysis and interval sequences.

- **Importance of pre-processing**: The time series abstraction methods used in the literature are almost always ad hoc and heuristic in nature. In chapter 2 we will show that successful knowledge discovery is then only possible if certain assumptions hold and that these assumptions are not fulfilled in general. Chapter 2 provides a survey of abstraction methods and shows that most of them are not suited for KDD purposes. We propose to use a method by Witkin [Witkin, 1983] and Bakshi [Bakshi and Stephanopoulos, 1995], which is the only method that satisfies our requirements but seems to be almost unknown in the data mining community. (Excerpts of chapter 2 have been published in [Höppner, 2002c, Höppner, 2002e].)

- **Efficiency of Pattern Discovery**: A new algorithm to discover all frequent interval patterns is proposed in chapter 3. The usefulness of such an approach has been recognized by some authors, but they do not provide an efficient solution (a similar goal has been formulated in [Bellazzi et al., 2002], where standard techniques are used to get an approximate solution, and also in [Kam and Fu, 2000], where the authors restrict themselves to a significantly reduced subspace and state that "discovering all possible patterns can be computationally inhibitive. [...] Experimental results show that even a small extension [...] would lead to a much increase in computational cost"). Chapter 3 provides the details of the algorithm and the evaluation section shows that the algorithms are very efficient. (Excerpts of chapter 3 have been published in [Höppner, 2001b, Höppner, 2001a, Höppner and Klawonn, 2002c].)

- **Qualitative vs. quantitative**: In the literature most of the approaches to learn from time series are either purely numerical (e.g. [Shao, 1998]) or purely symbolical (e.g. [Carrault et al., 2002]). The numerical approaches usually suffer from a lack of interpretability, the symbolic approaches have no means to refine symbols by quantitative constraints (and thus prior knowledge is frequently used to distinguish "increasing" segments from "highly increasing" segments). In this work, we start from an abstract, symbolic representation that is stepwise *refined* into a quantitative representation wherever this appears to be advantageous. Thereby we overcome the limitations

of the pure approaches. In chapter 4 new techniques for learning meaningful thresholds in the context of interval sequences are proposed. (Excerpts of chapter 4 have been published in [Höppner and Klawonn, 2001a, Höppner and Klawonn, 2002a].)

- **Ambiguity in perception:** Usually, it is extremely difficult to nail down a human to make precise statements when visual perception is involved. This is not only because experts seem to prefer being vague and imprecise in general, but also because human perception adapts pretty well to the context – a small increasing segment may be considered as important in some context, but neglected in some coarser view on the data. We are not aware of any symbolic machine learning techniques that respect this ambiguity in general or in time series perception in particular. In chapter 5 we discuss how this ambiguity is considered in the knowledge discovery process. (Excerpts of this chapter have been published in [Höppner, 2002b].)

- **Generalization:** It is well-known that a certain learning bias is absolutely necessary for a learning method to be successful [Mitchell, 1997] (if no bias is present, there is no rationale for generalizing beyond the observed cases). The bias in this approach is given by the space of patterns to be examined. However, it is also well-known that this bias may be the cause for missing some relationships in the data that cannot be represented by instances of the chosen pattern space. In chapter 6 a novel approach to overcome the "inadequacy of the pattern space" is proposed. (Excerpts of this chapter have been published in [Höppner, 2002a].)

Chapter 2

Time Series Abstraction

In this chapter we consider the problem of a meaningful subdivision of real-valued time series into similar subsequences. Similar subsequences will get the same label and thus become identical on a more abstract level of representation. Depending on the application area different notions of similarity may be appropriate, that is, different sets of attributes that characterize the time series may be chosen. As an example, in some cases it might be sufficient to distinguish increasing and decreasing trends, in other cases it might be important to distinguish between linearly, logarithmically, and exponentially increasing segments. In this chapter we will discuss several different techniques proposed in the literature. The goal of any of these techniques is to turn a continuous time series into a series of labeled intervals, where the labels address a property in the original series within the associated time interval. Later we want to use these labeled intervals as building blocks for more complex temporal patterns using Allen's temporal logic. Thus, the original time series will be discarded and any further analysis will rely on the validity of the abstractions. We will show, however, that most of the methods in the literature are not suited for generating valid abstractions, since they falsify properties of the original data, such as the position of extrema or the slope of the segments. Invalid abstractions naturally hinder the proper identification of interesting patterns. We will finally propose to use a method by Witkin and Bakshi that is sound, free of parameters and able to compensate the mentioned problems – a method that seems to be unknown in the data mining community.

Since we hope to discover knowledge from time series that can be presented to and easily understood by a human, we implicitly assume that we examine well-behaving signals only. We suppose that the signals are twice differentiable almost everywhere (we allow only for a small (finite) number of discontinuities in the signal or its first few derivatives). Besides that, we also assume that the signals have a finite number of extrema and inflection points and that the sampling rate has been chosen sufficiently high to detect them.

By a segment of a signal we understand a sequence of measurements being supported by the largest contiguous interval over which a certain property holds (e.g. increasing, below a certain threshold, etc.). Such a segment is labeled according to this property. It goes without special emphasis that domain specific knowledge about the time series, if present, should be used to guide any segmentation. Regarding a suitable algorithm to automatize this step, we require consistency (similar features in the time series lead to similar features in the representation) and robustness (minor changes, e.g. noise, should have little effect on the representation). Since the

steps in the subsequent chapters will be computationally quite expensive, a reduction of the amount of data is recommendable, that is, the average segment length should be clearly greater than the sampling rate.

The problem of time series partitioning can be considered either as a supervised or unsupervised task. It can be supervised in the sense that the attributes of interest are defined a priori and we have to assign labels from this given set to portions of the time series. Thus, in this deductive approach, we know in advance for what kind of features we are looking for. On the other hand, we can perform unsupervised partitioning in the sense that no such set of labels is given in advance and we thus have to learn this set from data, too. To attack both tasks we make use of a number of different machine learning and function approximation methods, which will be sketched only briefly in the following, since they are not in the centre of our interest here. We refer to the cited literature for more details. The inductive approach is discussed in the following section 2.1, the deductive approach in section 2.2. Section 2.3 summarizes which of these methods are considered as most promising and thus will actually be used for the segmentation in the remainder of this work.

We will demonstrate the results of some methods using the data set depicted in figure 2.1. We will refer to it as the ATTAS dataset.

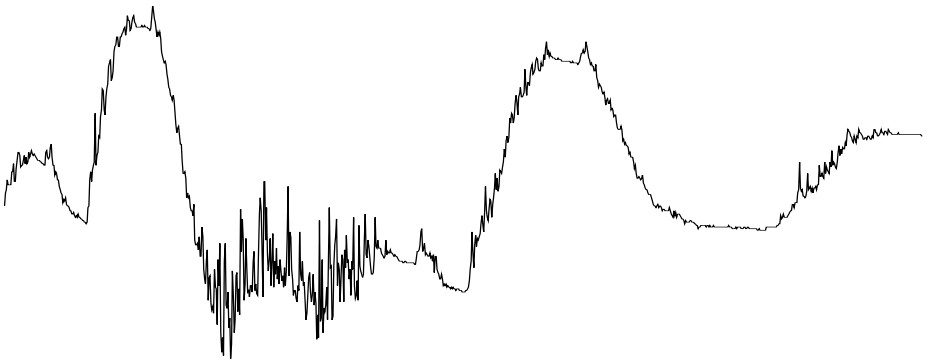


Figure 2.1: A test data set used in this chapter.

2.1 Inductively Derived Set of Labels

In this section we consider the problem of inductively deriving a set of labels and a segmentation of real-valued time series according to the induced labels. This can be done by identifying similar parts in the time series via clustering: clustering aims at partitioning data entities such that similar data objects belong to the same group and dissimilar data objects belong to different groups. If we think of small portions of time series as data objects, every cluster can be considered as an inductively derived label for a group of similar portions.

The crucial point in clustering is the used notion of (dis)similarity, which is implemented by means of a distance measure. A zero distance indicates a perfect match between two subseries, a large value indicates dissimilarity. Closely related is the question how to represent segments of the original series as new data entities, since the distance measure will be defined on this representation. Quite often, (parts of) time series are mapped to a high-dimensional space such that each a part of the time series corresponds to a point in the new space. In this approach, subseries of different length impose some additional problems, since all of them have to be

mapped into a space with a fixed dimension. At the end of the clustering process we obtain a "codebook" of representative shapes (one shape for each cluster) and may use the degree of matching to these "shape prototypes" for the segmentation. An expert of the field may also find linguistic descriptions for the clusters to better characterize the shape they represent in domain-specific terms.

Traditional clustering techniques partition a set of attribute vectors rather than portions of a time series. In the next section 2.1.1, we will therefore concentrate on different approaches to transform the given problem into another problem that is more appropriate for traditional clustering methods. These approaches distinguish in the way in which the time series over a certain time interval is embedded into a single attribute vector, and how the distance between these attribute vectors is defined. In section 2.1.2 we then focus briefly on some algorithms that may be used for the clustering itself.

2.1.1 Learning Shapes by Clustering

2.1.1.1 Clustering Embedded Subsequences

Labeling short subsequences in a real-valued time series can be considered as a vector quantization task. A window of constant width Δt_{win} is slid along the series, the content of each window is transformed into a tuple of observations. If data has been measured uniformly over time, this technique embeds a constant number of n consecutive values into an n-dimensional vector of observations. A series of length N gives us $N-n+1$ such vectors. This technique introduces a user-defined parameter n but allows us to use conventional clustering or vector quantization algorithms for subsequence clustering. It is a popular method used by many other authors (e.g. [Das et al., 1998, Geva, 1999, Karimi and Hamilton, 2000, Lin et al., 2000]).

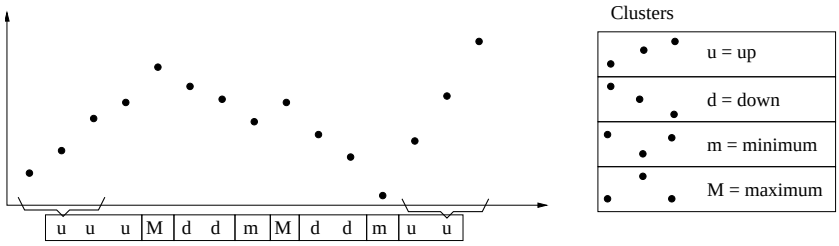


Figure 2.2: Three consecutive data objects are considered as a triple and the set of triples is clustered. The resulting prototypes are given on the right, the labels for each triple are given below the time series. Sequences of identical labels form an interval sequence.

Many clustering algorithms use the squared Euclidean distance as the dissimilarity metric. This pointwise comparison yields low distances only if one segment is almost an exact copy of the other. However, the shape of the subsequence is the main factor in perceived similarity rather than an exact match of the values. Two series may have the same shape although they have different baselines, for instance. Therefore, the attribute vector may be transformed, such as subtraction of the segment mean (to eliminate baseline effects) or normalization (zero mean and unit variance). In [Das et al., 1998] a greedy clustering algorithm is then used to form C subgroups of similar vectors.

Finally, the cluster c_i to which a segment belongs is used as a label for this datum. The real-valued time series $(t_i, x_i)_{1 \leq i \leq N}$, $x_i \in \mathbb{R}$, is thereby transformed into a

discrete series $(t_i, c_i)_{n \leq i \leq N}$, $c_i \in \{1, .., C\}$. While an x_i describes the value of the series at a single point in time t_i only, the cluster c_i also considers the history of $n-1$ observations in the past. We can transform this discrete time series into a labeled interval series by concatenating all consecutive elements $(t_i, c), ..., (t_j, c)$ with identical labels c ($\forall i \leq k \leq j : c_k = c$). The interval that is associated with this occurrence of c is given by the time points that denote the first and last element, that is $[t_i, t_j]$. Figure 2.2 shows the procedure schematically for $n = 3$. The resulting clusters are shown on the right, the segmentation at the bottom. While some authors indeed use $n = 3$ [Lin et al., 2000], the number of different clusters is very limited and the notion of increasing/decreasing segments is probably better captured by some of the approaches discussed later in section 2.2. For $n \gg 3$ the theoretical number of possible patterns that can be represented by the cluster prototypes increases rapidly.

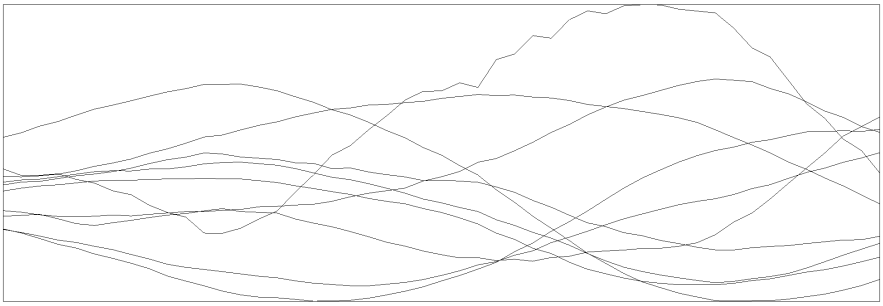


Figure 2.3: Resulting prototypes in one of the experiments (clustering 24h of embedded wind strength data, transformed to zero mean and unit variance).

However, with respect to our application the results of the conducted experiments were not very promising. Since the distance measure is not able to assign low distance values to dilated window contents, we obtain very short intervals $[t_i, t_j]$. Furthermore, only a fixed time interval of Δt_{win} is used to measure the similarity, quite different clusters may be found when using other window widths. In the experiments, the resulting clusters often appear very much like dilated and translated trigonometric functions with different amplitudes (cf. figure 2.3). Thus, a similar result can be obtained by using the first few Fourier coefficients to characterize the window content, which has the benefit of a reduced data dimension from n to 3 or 4. (This notion of similarity has been proposed in [Agrawal et al., 1993].) Besides n-tuples or trigonometric functions other models can also be used to represent the clusters. For instance, we may characterize the set of n observations associated with one cluster by a regression model. This has been done for the case of piecewise linear models in [Höppner and Klawonn, 2000a] and for a mixture of Gaussians in [Gaffney and Smyth, 1999]. Both approaches can deal with a varying number of observations per window, that is, with a varying number of data points per window, but cannot solve the problems of time-scaling, need a fixed window width and have higher computational needs.

2.1.1.2 Clustering of Embedded Models

With respect to the inadequate handling of time-scaling, it seems more promising to embed a more abstract representation of the series rather than the raw data. For instance n segments of a piecewise linear representation can be characterized by a $2n$ vector consisting of slope and length of each segment. Then, clustering is performed

on these "embedded models" rather than the embedded series. The advantage of this approach is that it handles translation and dilatation of the time series to some extent: A deviation of a prototype in the even components corresponds to a local shortening or lengthening of the profile, a deviation in an odd component corresponds to a scaling in the vertical axis. By using gradient information rather than absolute values we do not only compensate for different baselines in the segments, but address local differences in shape: If two segments distinguish only in a different slope at the beginning, but then behave structurally identical, their pointwise distance will be very high since the initial difference in slope separated both profiles. However, the pointwise distance of their derivatives yields much smaller distance values, since the derivatives differ only at the beginning. In this example, the slope values of the piecewise linear approximation served as an estimate of the derivative.

This has been done with $n = 3$ and the piecewise linear approximation shown in figure 2.10. The fuzzy c-means clustering algorithm has been used to obtain the resulting prototypes in figure 2.4. Note that the prototypes have different temporal extension. Thus, the problem of a fixed window width, which we have discussed in the previous section, is somewhat relaxed by this technique, since the segments in the original series are allowed to have different temporal extension.

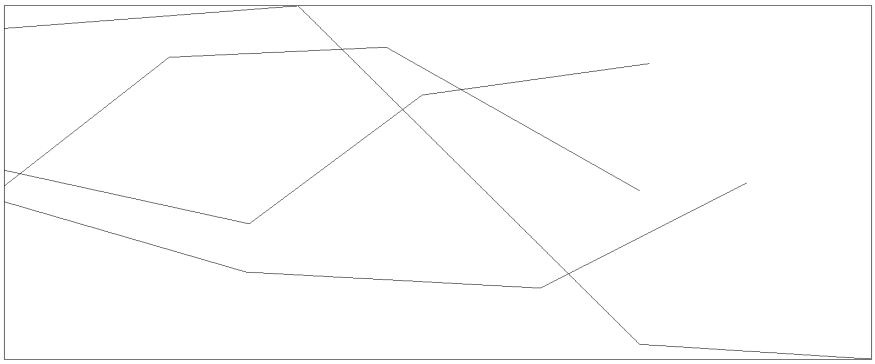


Figure 2.4: Resulting prototypes when using embedded linear segments.

Compared to the embedding of raw data, embedding higher representations appears much more promising. It can be refined further by selecting a different clustering algorithm that allows for a scaling of each component in each cluster (e.g. [Keller and Klawonn, 2000, Klawonn and Kruse, 1995]). Then, the sensitivity of each slope/duration value is also determined, providing additional information about which portions of the pattern is stretched within the cluster.

The kind of shape extraction proposed in this section can be considered as a very special case of the kind of temporal patterns we will search for in the next chapter.

2.1.1.3 Clustering by Warping Costs

We have seen that point-to-point similarity gives bad estimates of structural similarity if there is expansion or contraction in time. If other effects like vertical scaling are not significant, dynamic time-warping (DTW) can be used to locally shrink and stretch the time axis such that a point-to-point similarity of two time series is minimized [Sankoff and Kruskal, 1983, Berndt and Clifford, 1996]. While DTW can be used to compare series of different length, vertical distortions in the function values tend to be compensated by time warping even if there is no temporal dilata-

tion present. To compensate this undesired behaviour, Keogh proposes to warp the derivatives rather than the original function in [Keogh and Pazzani, 2001].

To make use of DTW for our problem, given n series an $n \times n$ symmetric dissimilarity matrix may be calculated, where $D_{i,j}$ denotes the warping cost of warping time series $\#i$ to series $\#j$. Then, relational clustering algorithms can be used to cluster the sequences into homogenous groups [Jain and Dubes, 1988, Duda and Hart, 1973]. The series in figure 1.1 will be clustered correctly by this method.

The approach is useful if the number of series to be compared is quite small and possibly of varying length. If we have a long, multivariate time series, the generation of all subsequences of a given length is prohibitive, since this would lead to a very large dissimilarity matrix. Therefore it is a good idea to use subsampling to reduce run-time requirements, as it is done in [Oates, 1999] to identify distinctive subsequences in time series.

If we use the data embedding discussed in section 2.1.1.1 to generate the subseries to be warped, this approach also suffers from the fixed window width (for two series to be similar they have to be similar *in the whole window of n observations*). But again it is possible to use the kind of embedding discussed in section 2.1.1.2 and then adapt dynamic time warping to the embedded models. For the case of linear segments this has been done in [Keogh and Pazzani, 1999b].

2.1.1.4 Clustering using Markov Models

Another approach is to learn a hidden Markov model (HMM) for each subsequence [Smyth, 1997] and to cluster the resulting models. HMMs assume that there is a fixed number of hidden (unobservable) states in which a system may currently be. The output of the system depends solely on the current state. Learning an HMM corresponds to estimating the state transition probabilities and state output probabilities via likelihood maximization. While this is a promising approach for long sequences, it may fail with short subsequences (windows) since they contain only little data to learn the high number of HMM parameters robustly. The state transition probabilities indirectly correspond to state durations and thus can be used to model time dilatation. HMMs are used in speech recognition, replacing DTW techniques that have been used in this domain before the advent of HMMs.

Instead of HMMs Markov chains can be learned from each subsequence. Since Markov chains are simpler than HMMs (less parameters, no hidden states) they can also be learned from shorter subsequences. However, their expressive power is limited compared to HMMs. They have been used in [Sebastiani et al., 1999] to cluster discrete-valued time series.

One advantage of both approaches is that it is possible to cluster subsequences of different lengths. After learning the model (HMM or Markov chain), each subsequence is characterized by the parameters of the model. If we choose the same model for all subsequences, we obtain a constant number of parameters for each subsequence. The probability that a given series has been generated by a given HMM is used to define the distance between time series and models. Difficulties arise from the fact that we have to select a single model structure that is adequate for all clusters (same structure and number of hidden states).

2.1.2 Suitable Clustering Algorithms

All techniques that use embedded series blow up the data volume. For instance, embedding n consecutive data points into a vector increases the data size from N (the time series alone) to $n \cdot (N - n + 1)$ (the $N - n + 1$ vectors of size n). But even without this blow-up, the database size might already be considerably large. Therefore an appropriate clustering algorithm that can handle a large data volume has to be chosen.

There is a variety of clustering algorithms in the literature that have been designed explicitly to cope with large data sets. In [Ester et al., 1996, Xiaowei et al., 1998] density-based clustering algorithms are proposed that seek for connected regions of high data density. These methods do not make any assumptions on the shape of the clusters and thus may detect clusters of arbitrary shape. But they require some thresholds to be fixed, regarding data connectivity and density. The subjective usefulness of the results may depend strongly on a correct setting of these parameters.

Also conventional clustering algorithms like Gaussian mixture decomposition or K-Means can be applied. They assume that data points are distributed according to some model (within an hyperball, according to some Gaussian distribution, etc.). For Gaussian mixture decomposition the expectation maximization algorithm is modified to scale with large datasets in [Bradley et al., 1999]. In their approach, whenever the main memory is filled with new data, the most similar data objects are summarized statistically and the summarization is used to carry on in further iterations.

For the fuzzy variant of K-Means a hierarchical data organization can be used [Höppner, 2002d] to gain considerable speed-up. Since with K-Means or Fuzzy c-Means the membership degree of a data object to a cluster prototypes depends on the distance to the prototype only, information about the relative positions (captured in the hierarchical data organization) can be used to estimate the range of possible membership degrees. It turns out that the membership degree is determined by the cluster centres in the vicinity of the data object, such that most of the cluster centres do not have to be considered in the calculation of an approximate membership degree. Once the membership precision is sufficiently high, this membership degree is used for all similar data objects.

Another popular approach that can be used for clustering (and visualization) are self-organizing maps (SOM) [Flexer, 1999]. Most often the data is mapped onto a two-dimensional plane, such that similar data vectors are mapped to neighboured points on the map. (Thus, SOMs are somewhat related to multidimensional scaling techniques.) SOMs have been used for clustering multivariate temporal features in [Guimarães and Ultsch, 1999], or newsgroup articles in [Honkela et al., 1997], for instance. Since the map is generated using gradient descent methods, it may take considerable time to learn such a map.

In some occasions a greedy clustering algorithm might be sufficient, which simply aggregates all data objects within an a priori fixed range around the cluster prototypes. Whenever a data object is encountered that is not sufficiently close to any of the already fixed cluster centres, it serves as a new centre. This approach has been reported to be useful in [Das et al., 1998]. A reason why this simple approach may yield superior results compared to more sophisticated algorithms is the fact that the number of elements per cluster is probably pretty small when Euclidean distance is used. In case of an embedding as discussed in section 2.1.1.1, (pointwise) similar shapes are comparatively infrequent. While we have only a very limited number of

local extrema patterns in the ATTAS time series (figure 2.1), there are much more dissimilar segments between the extrema. If we choose the number of clusters k too small, algorithms like K-Means tend to lump data of different clusters together, leading to inferior results compared to the greedy approach, where no specification of k is necessary. On the other hand, such an algorithm ends up with a very high number of clusters.

None of the clustering algorithms is free of parameters, k-Means and variants, Gaussian mixture decomposition, and Hidden Markov Models need the number of clusters/models k, self-organizing maps need the size and dimension of the map to be learned, and the greedy and density-based algorithms require the beforementioned thresholds on connectivity and/or minimum density. Learning the optimal k requires additional computational effort.

2.2 Deductively Derived Set of Labels

Among the most frequently used shape descriptions for time series in technical systems are terms like "linearly increasing", "convexly decreasing", or "constant" (see for instance [Bakshi and Stephanopoulos, 1995, Konstantinov and Yoshida, 1992]). There are 7 basic shapes for describing local trends in a function f, corresponding to the 3^2 possible combinations of positive/zero/negative first and second derivative f' and f'', excluding those 2 cases of zero f' and non-zero f'' which may hold only for a single point (extrema), but not for a whole segment. For time intervals of non-zero length we thus obtain the following basic shape descriptors: "constant" ($f' = f'' = 0$), "linearly increasing" ($f' > 0, f'' = 0$), "linearly decreasing" ($f' < 0, f'' = 0$), "convexly increasing" ($f' > 0, f'' < 0$), "convexly decreasing" ($f' < 0, f'' < 0$), "concavely increasing" ($f' > 0, f'' < 0$), and "concavely decreasing" ($f' < 0, f'' > 0$), as depicted in figure 2.5. What makes this representation attractive is that "a description that characterizes a signal by its extrema and those of the first few derivatives is a qualitative description of exactly the kind we were taught to use in elementary calculus to *sketch* a function" [Witkin, 1983].

If we want to use these basic shapes to describe the time series locally, breaking down the series into subsequences is – at least theoretically – straightforward: Via differencing we obtain estimates of the first and second derivative. Whenever one of the derivatives falls into a different subset of the partition $\{(-\infty, -\varepsilon), [-\varepsilon, \varepsilon], (\varepsilon, \infty)\}$ than its predecessor, we introduce a new segment. However, in almost every application the data is noisy. Noise makes the series oscillate around the true profile, thereby introducing a large number of tiny segments and local extrema. This highly fragmented representation does not correspond to the human perception of the time series. The main problem when using this approach is therefore how to compensate the influence of noise or, putting it differently, how to distinguish noise from significant features.

There are at least two different ways how to proceed:

- We can use function approximation techniques and extract the description of the time series from the approximating function instead of the original series. In this approach the problem of handling noise is handed over to the applied regression technique.
- We can use appropriate smoothing techniques to get more robust estimates of the first and second derivative and may increase ε. Then, handling noise correctly corresponds to selecting an appropriate smoothing filter.

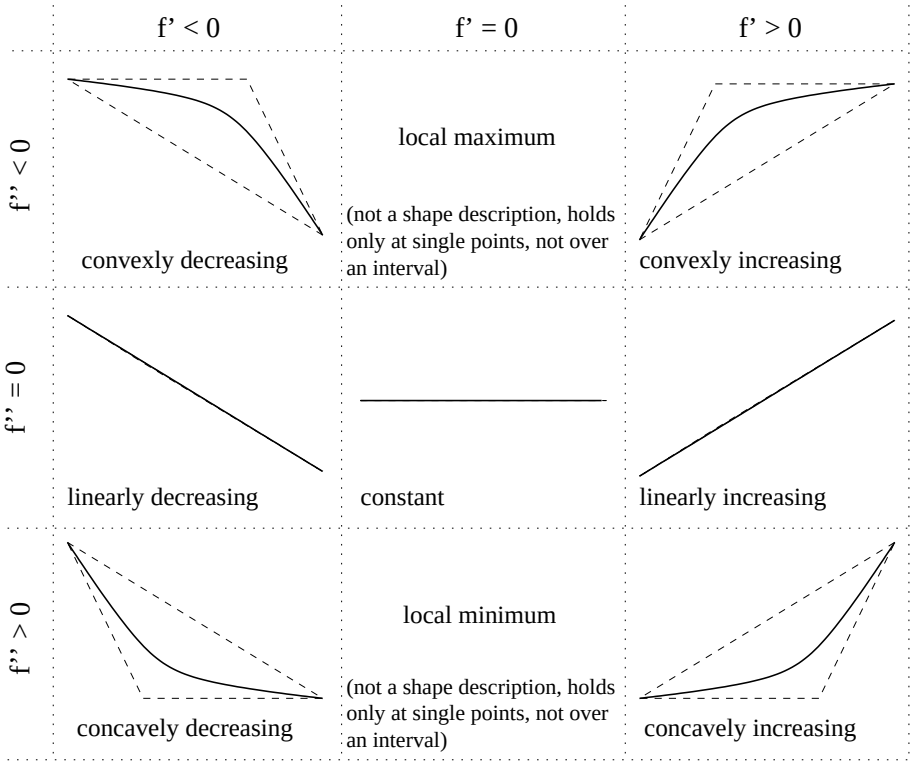


Figure 2.5: 7 basic shapes.

In the following sections, we will discuss methods for both approaches and their benefits and drawbacks.

2.2.1 Extracting Shape Information via Function Approximation

There is a large variety of different methods for function approximation or regression (see e.g. [Cherkassky and Mulier, 1998]). Having approximated the data by one of these methods we want to use the fitted function to extract shape information. For this approach making sense we require the following property: Any shape feature extracted from the approximating function must be supported by the original data. We will see that this property does not hold for all approximation techniques. Among competitive candidate methods we are especially interested in those methods that scale well with an increasing length of the time series and do not require much background knowledge to determine a suitable parameter configuration.

Almost all function approximation techniques use a combination of k basis functions b_i with weights w_i to define the approximating function $f(x) = \sum_{i=1}^{k} w_i \cdot b_i(x)$. Depending on a metric d to measure the pointwise error, the sum of residual errors

$$E = \sum_{j=1}^{n} d(f(x_j), \hat{x}_j) = \sum_{j=1}^{n} d\left(\sum_{i=1}^{k} w_i \cdot b_i(x_j), \hat{x}_j\right) \tag{2.1}$$

is often minimized in order to get a good approximation of the data set $\{(x_j, \hat{x}_j) | 1 \leq j \leq n\}$. We distinguish adaptive and non-adaptive function approximation methods, the latter addresses those techniques where the basis functions b_i are fixed

(unparameterized) and not subject to optimization. Since the only parameters are the weights w_i, the data is fitted to a linear combination of k basis functions b_i. Polynomials, splines and trigonometric functions (Fourier analysis) are prominent candidates for non-adaptive basis functions. With adaptive methods the basis functions have parameters which are also optimized. Usually, adaptive methods use much lower k than non-adaptive methods.

Once the basis functions are given, a linear combination thereof can be analyzed analytically. For instance, when considering the class of polynomials taking derivatives is a trivial task and standard algorithms can be used to find the zero crossings. Having approximated the time series (or a subsequence thereof) by a polynomial of appropriate degree, the zeroes of the polynomial yield the desired segmentation. With some methods, like neural networks for example, it might be painful to extract the first few derivatives analytically. In this case we may sample the learned function and use differencing to get estimates. Another approach, frequently used for noise reduction, is to cut off the high frequencies in the Fourier spectrum and use the reconstructed function in the time domain.

Difficulties lie in the choice of the degree of the polynomial or the cut-off frequency of the spectrum. But even worse, both techniques lead to analytical functions that have several local extrema or inflection points which are not supported by the original data. Figure 2.6 shows an example for the Fourier analysis and two short time series. Each series consists of two linear segments and are thus very similar. Despite the small differences in the input data, the smoothness of the approximation is different and various inflection points are introduced for the second example which have no correspondence in the linear segments. Setting small coefficients to zero strengthens these effects. The more the time series gets noisy, the more difficult it is to tell whether wiggling in the approximation is caused by the data itself or is introduced by the very nature of the used approximation method and its basis functions.

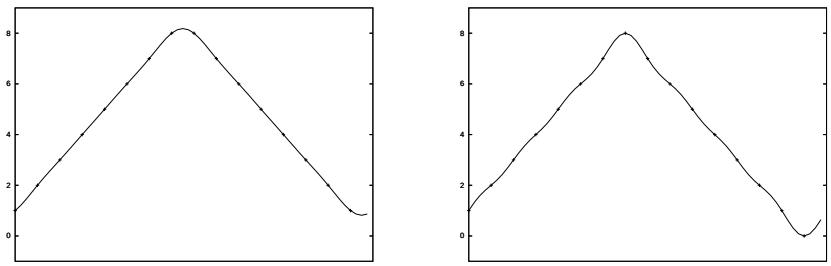


Figure 2.6: Fourier transformation introduces local trends (qualitative features) which are not supported by the data.

To overcome such difficulties it seems to be a good idea to require some constraints on the smoothness of the approximation. This can be achieved by reducing the local flexibility of the basis functions, for instance by using piecewise approximation with polynomials of low degree. To obtain good approximations with a moderate number of segments, piecewise methods do not use uniformly distributed break points but adapt the break points of the segments automatically (which turns the method into an adaptive method). Another approach is the introduction of penalty functions (see [Cherkassky and Mulier, 1998]) to favour smoother solutions. Unfortunately,

increasing the smoothness makes the detection of discontinuities almost impossible, since then discontinuities are extremely blurred.

A drawback of some methods like the Fourier transform is the infinite support of the basis functions: a slight change in one parameter (e.g. when setting small coefficients to zero) has global effects and may potentially introduce or shift zero crossings anywhere in the first or second derivative. We speak of *local approximation* if the support of the basis functions is finite.

In the following sections we will examine a small number of function approximation techniques with respect to the applicability to our problem. The simplicity of piecewise linear functions as well as its efficiency in representation motivates many authors to develop such approximation algorithms, some of them will be discussed in the next two sections. As a matter of fact, linear function do not have local extrema or inflection points. However, a sequence of increasing segments with decreasing slope values may be summarized as a single "convexly increasing" segment. Then a segmentation according to extrema and inflection points takes place at the end points of the linear segments. Later in section 2.2.1.3ff we will focus on other regression techniques, like splines and wavelets. Besides these methods, there are a lot of other methods like orthogonal polynomial approximation, radial basis function networks, neural networks, or support vector machines for regression. Some of them have prohibitive computational complexity for knowledge discovery purposes, others are very fast. However, despite the clear differences in the computational complexity, with respect to our purposes the benefits and drawbacks of these methods are similar to those of the methods that we will discuss briefly in the next sections.

2.2.1.1 Uniform Approximation

A time series or signal is approximated uniformly, if the approximating function f deviates from the original values by no more than a certain $\varepsilon > 0$, that is

$$\forall i : |f(x_i) - y_i| \leq \varepsilon. \tag{2.2}$$

In principle it is a good idea to require uniform approximation, since then the original data is approximated equally well over the complete range. Uniform approximation can be formulated as a minimization of ε under the condition $\|f(x_j) - \hat{x}_j\| < \varepsilon$, which can be solved using methods from linear optimization [Krabs, 1969]. However, the number of constraints is proportional to the number of data objects, which makes the computational effort to solve the problem quite high. Therefore, many approaches assume that the user has fixed the value of ε. However, the specification of ε might cause some serious problems, as we will see below. Among polynomial approximations, linear polynomials are the simplest (polynomials of zero degree provide no qualitative information about the profile). For the discussion in this section we are satisfied with a simple (non-optimal) uniform piecewise linear approximation algorithm.

Driven by growing demands in computer graphics, many segmentation algorithms have been developed, mainly for the conversion of bitmap images into vector graphics (e.g. [Pavlidis and Horowitz, 1974, Rosin and West, 1995]), but also for time series approximation. The so-called "on-line" algorithms read data point by data point and adjust a current linear segment with every new point. After the adjustment, it is checked whether the data points in the current segment still satisfy (2.2). If that is not the case, a new segment is introduced. This greedy method does not optimize anything but simply keeps (2.2) satisfied. However, this approach is appealing from a computational point of view. While in a naive approach

we would have to iterate over all points in the segment to check condition (2.2), in [Sklansky and Gonzalez, 1980] a more clever solution to this problem is proposed. The algorithm yields a subset of the time series points such that their piecewise linear connection satisfies (2.2). Starting from a point $\mathbf{x}_j$ in the series, a cone is initially constructed such that it touches the circle of radius ε around the next data point $\mathbf{x}_{j+1}$. The linear continuation from $\mathbf{x}_j$ must stay within this cone in order to guarantee a deviation below ε from $\mathbf{x}_{j+1}$ (see figure 2.7a). The same is true for all the following data points, that is, the intersection of cones of valid linear continuations becomes smaller with every new point (figure 2.7b). Once the line from the starting point to the next data point is not contained in the cone (figure 2.7c) the segment is finished and a new one is started. Using this technique, a single pass through the time series is sufficient and storage needs are very low.

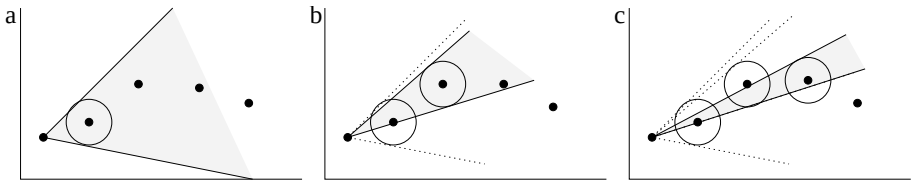


Figure 2.7: Uniform approximation algorithm [Sklansky and Gonzalez, 1980].

Figure 2.8 shows the results of the algorithm. If ε has been chosen appropriately, the results are very good (example in the middle). From left to right we have increased the noise but left ε unchanged. Now, uniform approximation yields different representations for the data, although the underlying function remained the same. If ε is chosen too high (left example), the approximation gets somewhat loose, if ε is chosen too low (right example), the noise gets modelled. Ideally, we would like to get *identical* representations.

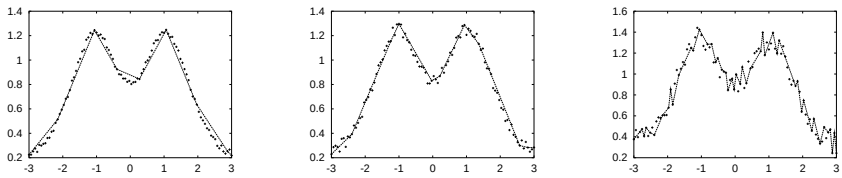


Figure 2.8: Uniform approximation with constant ε and varying noise.

In many knowledge discovery applications we cannot be sure that the amount of noise does not change over time. If the signal to noise ratio changes we will obtain (at least locally) unsatisfying results whatever ε we might choose. In [Goodrich, 1994] an algorithm for optimal uniform piecewise linear approximation with k segments is proposed that runs in $O(n \log n)$ time. Instead of ε the number of segments k has to be provided in advance, but its specification is almost as difficult as the choice of ε.

We can adapt heuristics to fix ε or k: If we have to fix ε we can try to estimate the noise or variance locally (see appendix A.3) and then enlarge the balls around the data points proportional to the standard deviation while using $\varepsilon = 1$. If we have to fix k we can start with an overestimation of k and merge segments that have similar values of slope. The merging operation reduces k and the algorithm is restarted. We stop if no more merging is possible. So we still have to fix a threshold on the slope similarity, but this parameter has a direct correspondence to the visual appearance and should be easier to fix.

2.2.1.2 Least-Squares Piecewise Linear Approximation

In this section we consider least-squares approaches to piecewise linear approximations (some of the techniques generalize easily to polynomials of higher degree).

Greedy Algorithm. As in our discussion in section 2.2.1.1 it is also possible to develop a greedy $O(n)$ segmentation/approximation algorithm for least-squares error: instead of (2.2) we require that the mean error per segment stays below a threshold ε. If the segment error exceeds ε we start a new segment. Using the last point of the previous segment as an anchor for the next segment, a continuous representation is obtained. In practice, similar problems as shown in figure 2.8 for the uniform approximation occur, due to the a priori specification of ε. The robustness of algorithms of the ε-kind is limited in the sense, that very small changes may be responsible for the introduction of new break points.

Bottom-Up/Top-Down Segmentation. In [Keogh, 1997] a method for obtaining a piecewise linear representation is proposed that does not require the specification of a maximal deviation ε in advance. The method starts with a subdivision of the whole series into small segments (of initial length 3). For each segment, a linear approximator f_j is fitted. Then, the residual error for each segment S_j is calculated

$$e_j = \frac{\sum_{(t,x)\in S_j}(f_j(t) - x)^2}{|S_j|}, \tag{2.3}$$

which is the sum of squared errors divided by the number of data points in the segment (mean squared error). In general, the approximation quality (error) will not be equally well for all segments. The overall regularity of the errors over k segments is captured by the standard deviation σ_k. Next, two neighbouring segments are merged such that the resulting σ_{k-1} of the new approximation is minimized. This process is repeated until k reaches 1. The partition with the lowest value of σ_k will finally be selected. This algorithm generalizes easily to polynomials of higher degree.

The underlying idea of the algorithm is to select the approximation where the error is balanced among the segments, based on the assumption that the noise level is constant over time. Keogh reported that the algorithm leads to very similar subdivisions when applied to the same time series with varying the amount of (stationary) noise. Although the idea of circumventing the specification of ε in advance is appealing, the algorithm did not perform very well in our experiments. We observed for some time series that the variance increases monotonically with the decrease of k, such that the algorithm selects a large number of segments. This may happen if the time series is comparatively smooth, each segment has a low error, and no noisy segments are present. Intuitively, the number of obtained segments is too high in this case. For other data, such as ATTAS, where the assumption on constant variance is not fulfilled, the algorithm finally selected a representation with very few segments – the high variance in the noisy regions forces the other segments to adapt to this variance level. Thus, by the assumption of equal variance the presence of noise causes a sloppy adaption to the series elsewhere to obtain a common variance level. Some improvement can be obtained by providing additional estimates of the local variance in the least squares approximation (see appendix A.3), but this does not solve the problem in general. The final subdivision points will be a subset of the initial subdivision into $N/3$ segments, which may cause undesirable effects if data is not sampled at a constant rate. Another disadvantage is the fact that the segments are not continuous, allowing for various similar increasing segments without decreasing segments in between.

(Scaled) Euclidean instead of Vertical Distance. We have already discussed

in section 1.1 that noise does not only lead to variance in the output values but also in the temporal axis. When fitting a line to data according to (2.3) only the vertical or pointwise error is considered. When comparing two signals by visual inspection, however, we usually use the distance of a data point $\mathbf{x}$ perpendicular to the curve. With piecewise linear functions this distance is easily calculated by $|(\mathbf{x}-\mathbf{p})^\top \mathbf{n}|$ where $\mathbf{p}$ is a point on the line and $\mathbf{n}$ is its normal vector (perpendicular to the line vector). If $\mathbf{n}$ is normalized ($\|\mathbf{n}\| = 1$), then this dot product yields the Euclidean distance of $\mathbf{x}$ to the line. If we have selected $k+1$ knot points $\mathbf{p}_i$ (amounting to k linear segments), we obtain normal vectors by rotating the line vectors $\mathbf{p}_{i+1} - \mathbf{p}_i = (p_{i+1} - p_i,\, \hat{p}_{i+1} - \hat{p}_i)^\top$ by 90 degrees: $\mathbf{n}_i = (-(\hat{p}_{i+1} - \hat{p}_i),\, p_{i+1} - p_i)^\top$. Rather than considering a normalized norm vector, we use the unnormalized $\mathbf{n}_i$ for the error estimation. This corresponds to a locally scaled error function, where the error in each segment is scaled by the length of the segment. It is obvious that this local scaling has no effect if all segments are approximately of the same length. If there are short and long segments, then the scaled error function handles short segments more flexible than long ones, since the sum of squared errors is multiplied by a larger number for the long segment. Thus in order to obtain a better fit to the data, we tolerate larger errors in small segments while putting special emphasis on small approximation errors with long segments. Thus we have the following scaled error function

$$E = \sum_{j=1}^{n} \sum_{i=1}^{k} (l_i \cdot |b_i(x_j) - \hat{x}_j|)^2 \tag{2.4}$$

where the factor l_i is the length of $\mathbf{n}_i$ and b_i is the linear function that connects the two knot points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ and is zero elsewhere. For given knot values p_i we obtain the output values $\hat{p}_i$ of a continuous piecewise linear approximation using (2.4) by solving a system of linear equations (see appendix A.1). Due to the finite support of the linear basis functions, the coefficient matrix is tridiagonal and can be solved in $O(k)$.

From figure 2.9 we can see, that there are always two ways how to improve the fit to the data. We have discussed the case of a vertical knot point adjustment, which is the usual case in non-adaptive approximation. Using (2.4) it is also possible to solve the system of linear equations for the knot points p_j instead of the knot values, thereby adjusting the length of the segments. This turns the method into an adaptive approximation method.

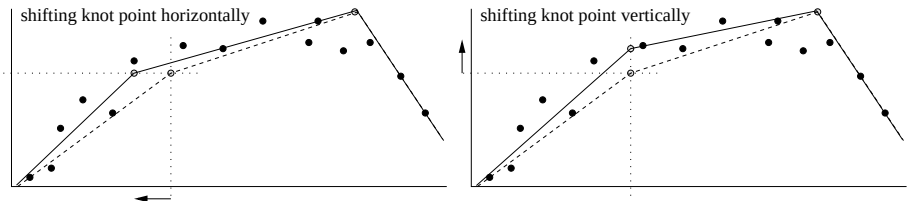


Figure 2.9: The fit of a piecewise linear approximation (dashed line) can be improved (solid line) by either shifting the knot points horizontally (left) or vertically (right).

The resulting algorithm ([Höppner, 2000], appendix A.1) alternatingly updates the output values at the knot points and the location of the knots points themselves. Each time a system of linear equations is constructed ($O(n)$) and solved (tridiagonal matrix, $O(k)$). Usually, a few iterations i are sufficient. The overall complexity is $O((n+k)\cdot i)$. A drawback of the algorithm is that the number of segments has to be specified in advance or has to be adjusted heuristically again. The final result in figure 2.10 for the ATTAS dataset has been obtained by starting with

an overestimated k and then removing one of two consecutive line segments with almost identical slope.

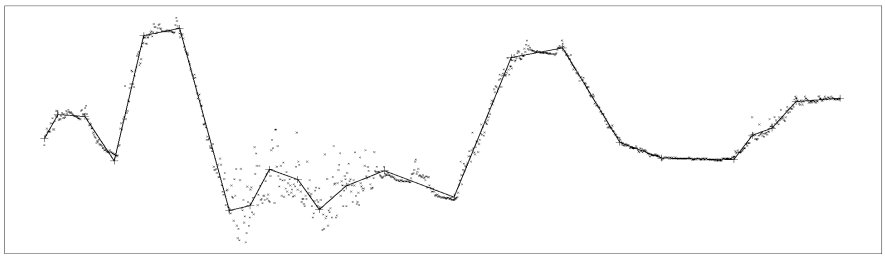


Figure 2.10: Approximation of ATTAS signal with piecewise linear segments.

The idea of a scaled error has been generalized from the univariate case to the multivariate case in [Höppner et al., 2002]. Another approach to fit piecewise linear functions using Newton's method can be found in [Julian et al., 1998].

2.2.1.3 Approximation by Mixture of Regression Models

The main problem with adaptive local approximation techniques is to find out where to break up the series into pieces or segments appropriately, such that the number of segments is kept minimal. In the identification of fuzzy systems one is also interested in describing input/output relationships by a preferably small number of local rules. There, the segmentation into rules is often considered as a clustering task, see e.g. [Chen et al., 1998, Höppner et al., 1999, Runkler and Bezdek, 1999, Yoshinari et al., 1993]. For our purposes of time series segmentation as well as for the finding of fuzzy rules we need a partitioning of the input axis only. However, simply clustering the input values is not a good idea – if the data has been measured uniformly over time we will obtain an equidistant partition of the time axis, which is not very meaningful. Thus the segmentation has to be guided by its suitability for approximation purposes. Fitting multiple polynomials (switching regression models) to a set of data has been done by [Hathaway and Bezdek, 1993]. However, in their approach data points that are approximated by the same regression model are not necessarily contiguous, which does not correspond to our notion of a time series segmentation. The key idea is the combination of both techniques, clustering of the input domain and regression in the input/output space. Once the time axis has been partitioned into segments, each model can be fitted to the data within the cluster easily. On the converse, a good approximation gives hints for an appropriate time segmentation. Thus, if segmentation (or clustering) and model fitting are coupled, the method adapts the segmentation to the goodness of fit. Since the approximation quality is directly influenced by the similarity of the data *and* the (fitted) regression model, the segmentation is guided by the shape of the series.

Thus, approximating a time series locally with simple functions can be decomposed into an iterative procedure consisting of segmentation and regression steps. Among the different types of clustering algorithms [Jain and Dubes, 1988], partitional algorithms are best suited for this purpose. The most prominent partitional algorithm is the k-means algorithm. Its fuzzy counterpart has been reported to be more robust in [Dunn, 1973], we therefore combined fuzzy c-means[1] and fuzzy c-regression models to get interpretable fuzzy models in [Höppner and Klawonn, 2000b].

[1] We use fuzzy algorithms, however, at the end we need a crisp segmentation. Usually the degree of fuzziness can be controlled by means of a fuzzifier to adapt to the needs of a crisp segmentation,

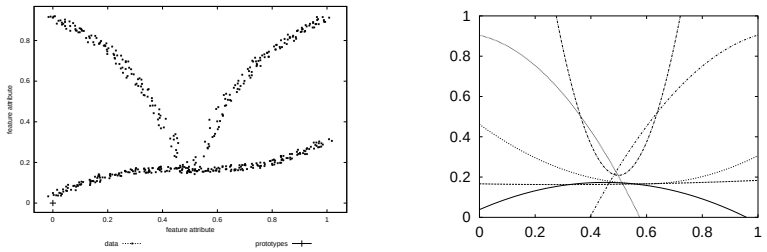


Figure 2.11: Left figure: Two noisy time series. Right: Adaptively fitted local polynomials of degree 2.

Figure 2.11 shows the result of the algorithm when approximating the two functions f and g on the left using $c = 3$ segments. Although it is not shown which polynomial belongs to which function one can guess it easily from the figure. Of course, polynomials of degree 2 cannot match the original functions (locally) perfect, but they catch the behaviour or shape of the function very well and are therefore valuable for shape extraction.

Unfortunately, the space and time complexity of this approach is comparatively high due to its iterative nature, therefore it is not very well suited for long time series in KDD domains. Since we perform local approximation it is not necessary to fit the whole series in one step. We could move a sliding window along the series, removing the leftmost segment from the current data set after each approximation and reading a few more data sets before proceeding with the next window. A drawback of this approach is that we need to specify the number k of regression models in advance.

A somewhat related approach is described in [Medino-Chico et al., 2001], where a regression tree is learned initially from data. The tree performs a greedy segmentation of the input space to define local (linear) models for each of the regions in the partition. Then, a neural network architecture is derived from the tree structure, thereby transferring the learned segmentation into a priori knowledge about the structure of a neural network. The neural network finally optimizes the approximation as well as the segmentation.

2.2.1.4 Splines

A spline [de Boor, 1978] is a series of locally defined low order polynomials. Splines were originally developed to smoothly interpolate between points, where constraints on the continuity (also on higher-order derivatives) are defined to ensure a smooth curve. Among all possible basis functions the cubic B-spline is most appealing, since it is simple, symmetric, and has continuous first and second derivatives. The family of B-splines (cf. figure 2.12) is defined recursively as

$$B_j(t) = \frac{1}{j}(tB_{j-1}(t) + (j + 1 - t)B_{j-1}(t - 1))$$

for $j > 1$ and $B_0 = \chi_{[0,1]}$. Often the B-splines are shifted to be centered around zero. Then, B-spline B_j is obtained from convolving $B_0 \equiv \chi_{[-1/2,1/2]}$ for $j+1$ times

see [Höppner et al., 1999]. A different approach is discussed in [Höppner and Klawonn, 2001b].

with itself.

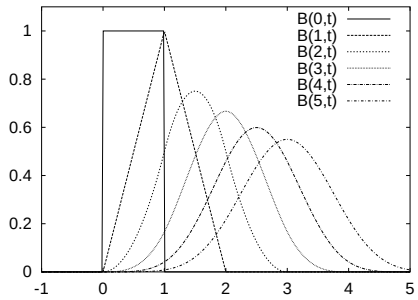


Figure 2.12: Various B-splines.

Between any two knots 4 cubic polynomials influence the function value of a cubic spline. For each of these intervals, the sum of 4 cubic splines is also a polynomial of degree 3, thus the zero crossings of the first and second derivatives can be computed in closed form and lead to the desired segmentation of the signal. Approximation using splines via least-squares and fixed knots is discussed in [de Boor and Rice, 1968a]. As before, the problem is how to select the number of knots (and their location) appropriately: The more knots we use, the more likely it is that we start to model the noise in the signal.

Several methods have been proposed to select knot positions appropriately, thereby turning splines into an adaptive method. In [de Boor and Rice, 1968b] they iterate over all knots and reduce the error by Newton's method, considering only a single knot at a time. In [Karczewicz et al., 1995] B-splines are used to interpolate the data first before knot points are successively removed such that the increase of the approximation error is kept minimal. Automated curve fitting with Bezier curves is described in [Schneider, 1990]. Since splines assume continuity we can expect problems when trying to approximate discontinuous time series, as reported in [Shatkay, 1995]. The multivariate adaptive regression splines (MARS) method [Friedman, 1991] adopts ideas from greedy recursive partitioning regression (CART) [Breiman et al., 1984] to locate the spline knot points. Figure 2.13 shows a spline approximation for the ATTAS data using a roughness penalty approach [Ramsay and Silverman, 1997].

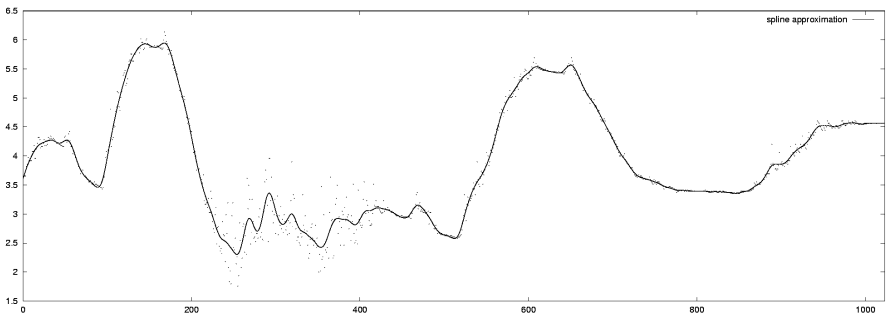


Figure 2.13: Cubic spline approximation.

Recently, the strong relationship to wavelet analysis (next section) and multiscale analysis (section 2.2.3) has been outlined [Chui, 1992, Wang and Lee, 1998]. We will revisit splines in section 2.2.3.4.

2.2.1.5 Wavelets

In contrast to sine waves in the Fourier transform, in the wavelet transform [Mallat, 2001] the basis functions have compact support in frequency *and* time[2]. Figure 2.14 shows some prominent wavelets. The Daubechies-20 wavelet, for instance, still exhibits the shape of a wave, but is limited in its temporal extension, which is where the name wavelet comes from. Since wavelets are local in time and frequency they are especially well-suited to characterize local phenomena in time series.

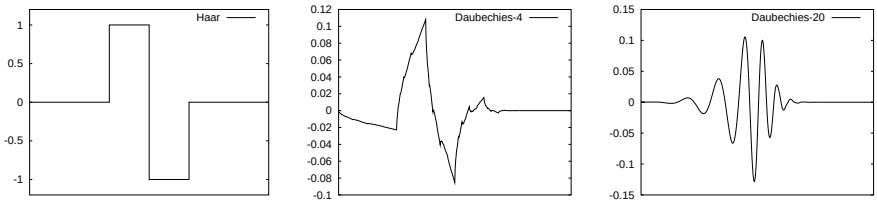


Figure 2.14: Some prominent wavelets (Haar, Daubechies-4, Daubechies-20).

Due to their locality wavelets have to be translated in time in order to approximate the whole series. A so-called mother wavelet[3] ψ is dilated and translated by $s \in \mathbb{R}$ and $u \in \mathbb{R}$ respectively, leading to a family of functions

$$\psi_{s,u}(t) = \frac{1}{\sqrt{s}}\,\psi\left(\frac{t-u}{s}\right)$$

The wavelet transform of f at time u and scale s is

$$W_{s,u}f = \langle f, \psi_{s,u}\rangle = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right)dt$$

The original function can be reconstructed from the wavelet transforms by

$$f(t) = \int_{s=-\infty}^{\infty}\int_{u=-\infty}^{\infty} W_{s,u}f\;\psi_{s,u}(t)\;du\,ds$$

For the fast discrete wavelet transform the dilatation and translation parameters are discretized dyadically as

$$s = 2^m, \qquad u = 2^m k$$

where m and k are integers and represent the dilatation and translation parameters. The translation parameter determines the location of the wavelet in the time domain, while the dilatation parameter locates the frequency range. Under some conditions the family of functions $\sqrt{2^{-m}}\psi(2^{-m}x - k)$, $(m,k) \in \mathbb{Z}^2$, forms an orthonormal basis of $L^2(\mathbb{R})$. The discovery of compactly supported wavelets ψ led to the finite discrete dyadic wavelet transform, which is $O(n)$ and thus even faster than the fast Fourier transform.

To obtain a smoothed representation of a time series, relatively small wavelet coefficients are set to zero, just as we have done with the Fourier coefficients. Due

[2] or at least decay rapidly

[3] A wavelet ψ is a function $\psi \in L^2(\mathbb{R})$ (e.g. $\int_{-\infty}^{\infty}|\psi(t)|^2dt < \infty$), which has zero mean, is normalized ($\|\psi\| = 1$) and centered in the neighbourhood of 0. The translated and dilated versions are also normalized.

to the locality of the wavelets, deleting coefficients does not affect the whole reconstructed time series any longer. But since we are interested in determining the zero crossings or extrema in the signal, the wavelets should be smooth (unlike the Haar or Daubechies-4 wavelet) and should not have too many zero-crossings itself (unlike the Daubechies-4 or Daubechies-20 wavelet). A 70% energy reconstruction of the ATTAS time series as depicted in figure 2.15 using the Daubechies-4 wavelet exhibits a rough profile with many local extrema and inflection points that are not supported by the data itself. We will discuss a more appropriate wavelet later in section 2.2.3.4.

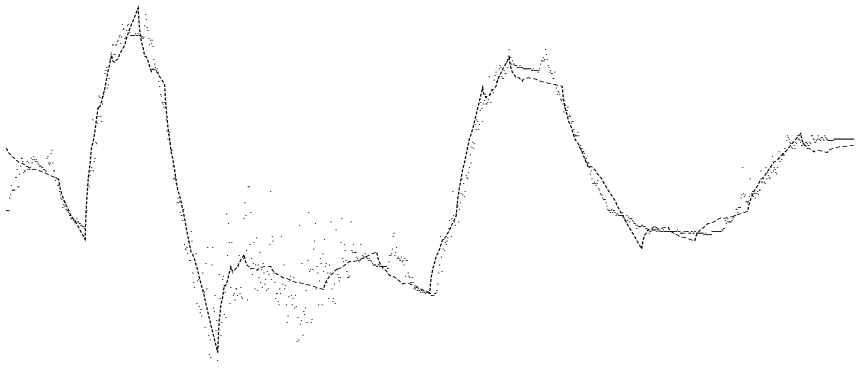


Figure 2.15: Reconstruction of the ATTAS signal using the Daubechies-4 wavelet.

2.2.2 Extracting Shape Information via Smoothing

In this section, we want to use smoothing techniques to compensate noise before trying to extract the qualitative description of the time series. For a smoothing method to make any sense, the value of the function estimate at time t must be influenced mostly by the observations near t, however, not all neighbours have to be considered necessarily to the same degree. The simplest and classic case of a smoothing estimator is the *kernel estimator*, where the estimate is given as a linear combination of the neighbouring observations. A kernel function $k(u)$ (also called filter or smoothing function) has most of its mass concentrated close to 0, and either decays rapidly or disappears entirely for $|u| \geq 1$. Common kernel functions are the uniform, the quadratic, and the Gaussian kernel [Ramsay and Silverman, 1997], as depicted in figure 2.16. The degree of smoothing is controlled by the scale or bandwidth parameter s that defines a dilated kernel $k_s(u) = k(\frac{u}{s})$.

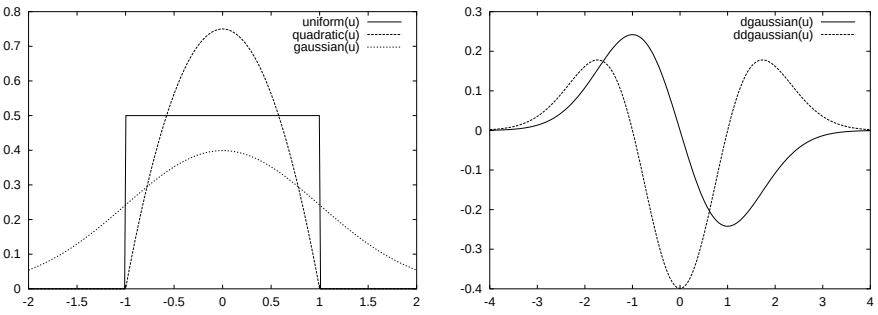


Figure 2.16: Kernel functions for smoothing (left) and derivative estimation (right).

The smoothed estimate $\bar{f}$ of a function f is obtained by convolving the original signal with the kernel function k_s, that is in the continuous case

$$\bar{f}(t) = (k_s * f)(t) = \int_{u=-\infty}^{\infty} k_s(u) \cdot f(t-u)\, \mathrm{d}u \quad \text{resp.} \quad \bar{f}[t] = \sum_{u=-\infty}^{\infty} k_s[u] \cdot f[t-u]$$

in the discrete case. Since we are interested in estimates of the first few derivatives of f, the following property (obtained from partial integration) is very helpful

$$(k_s * f')(t) = (k_s' * f)(t) \tag{2.5}$$

This means that we can simply convolve the original series f with the derivative of the kernel $\frac{d^n}{dt^n} k_s$ to obtain estimates of $\frac{d^n}{dt^n} f$. Other kernels to estimate derivatives (e.g. Gasser-Müller kernels) can be found in [Ramsay and Silverman, 1997].

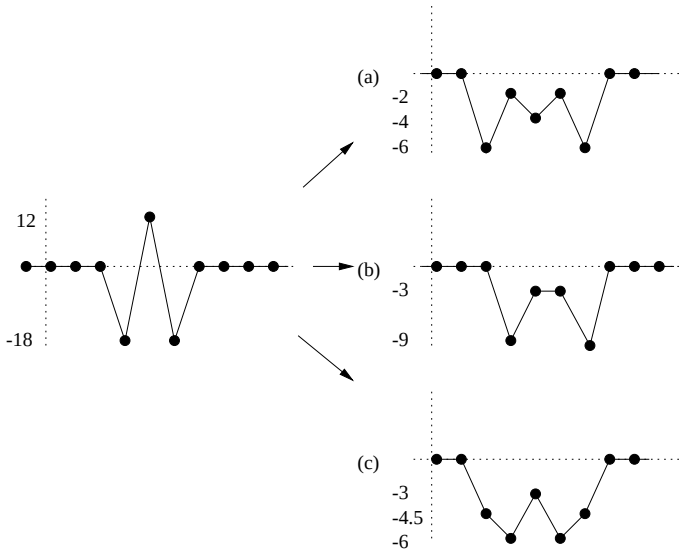


Figure 2.17: Application of various filters to the signal on the left.

Having obtained estimates of the first and second derivative we can easily scan for the zero crossings. From the signs of the first and second derivatives within each interval we obtain the qualitative behaviour. But which kernel function should we choose? When discussing the usage of function approximation techniques for the extraction of qualitative descriptions of time series in section 2.2.1, a fundamental requirement was that no qualitative features will be introduced by the function approximation technique without supporting evidence in the data. For some techniques we have seen that this may happen quite easily. The same requirement remains valid for the filter to be applied. Figure 2.17 shows a simple discrete time series on the left side, and a number of "smoothed" series on the right side, where the uniform filter of size 3 (filter coefficients $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$) and 2 (coefficients $(\frac{1}{2}, \frac{1}{2})$) have been used for series (a) and (b) and the filter $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ for (c). While the last two filters preserved the number of extrema, the "smoothed" signal (a) has 5 (compared to 3 extrema in the original series). This is somewhat surprising, because the intention of performing kernel smoothing was to get a smoother curve – we expect the number of extrema to decrease, not to increase. But as the example shows, some filters can easily insert qualitative features that are not supported by the original data.

We speak of a scale-space kernel k if the number of extrema of $k * f$ does not exceed the number of extrema in f. Fortunately it is possible to derive conditions

on the filter guaranteeing this property. In the continuous case the Gaussian kernel is the *only* kernel that satisfies this requirement [Babaud et al., 1986]. For a characterization in the discrete case see [Lindeberg, 1994].

The difficulties with this approach lie in the selection of the scale s denoting the size of the filter (or variance of the Gaussian). Smoothing always smears out not only noise but also original features of the time series – therefore the selection of the scale parameter appears as difficult as the selection of the number of knots k or threshold ε before. Another problem is the fact that the higher the scale s the more the features may be dislocated in the smoothed series. As an example, the arrow in figure 2.18 indicates a shift in the location of a local minimum due to smoothing. If these locations are used to extract durations of qualitative trends in the series, the estimated duration will not be exact – making it more difficult to find rules that depend on this kind of features.

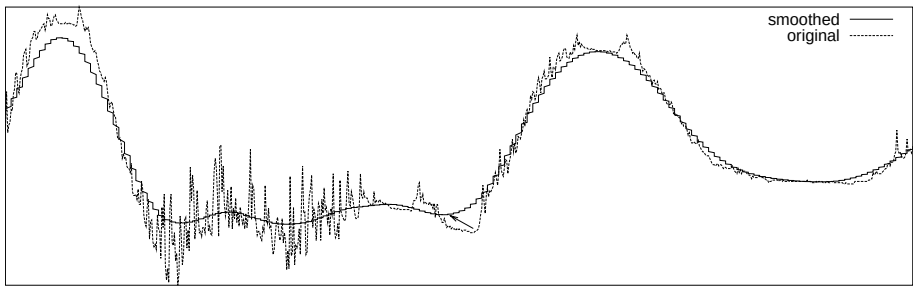


Figure 2.18: Smoothing dislocates features in the time series.

2.2.3 Multiscale Characterization

A typical signal from a process may contain contributions from a variety of sources, such as sensor noise, disturbances and faults [Bakshi, 1999] (cf. figure 2.19). Thus, the signal we analyse is a composition of a number of events with distinguished behaviour in time and frequency domain. Sensor noise is usually of high frequency and stationary (independent of time), faults cause sudden changes (local in time, high frequency event), equipment degradation or disturbances are usually of lower frequency. This makes it difficult for the smoothing approach to select an appropriate *scale* (or variance of the Gaussian), which determines how many neighbouring observations are incorporated to smooth the current observation. If we choose a large scale high frequency components will vanish (not only noise, but also sudden changes), if we choose a low scale we will have difficulties in recognizing more global trends in all the noise. Function approximation techniques usually do not better, since they have a somewhat related problem: quite often the basis functions stem from a particular class of equally smooth functions: using equally spaced cubic splines makes it difficult to capture rare events of high frequency and slowly changing trends at the same time. If we want to get rid of noise we may select smooth basis functions, but thereby we loose the possibility of modelling high frequency events (like 2.19(d)) appropriately.

However, humans are usually quite good in extracting all these different types of events from a time series profile, so how does a human distinguish between (high-frequency) noise and (high-frequency) features? Humans attention is clearly attracted by transients and changes as opposed to stationary stimuli, which humans

soon ignore [Mallat, 2001]. Humans automatically adapt the size of the neighbourhood that must be taken into account to distinguish local changes from stationary noise. A successful distinction is only possible, if *multiple* sizes of the neighbourhood are considered *at the same time*.

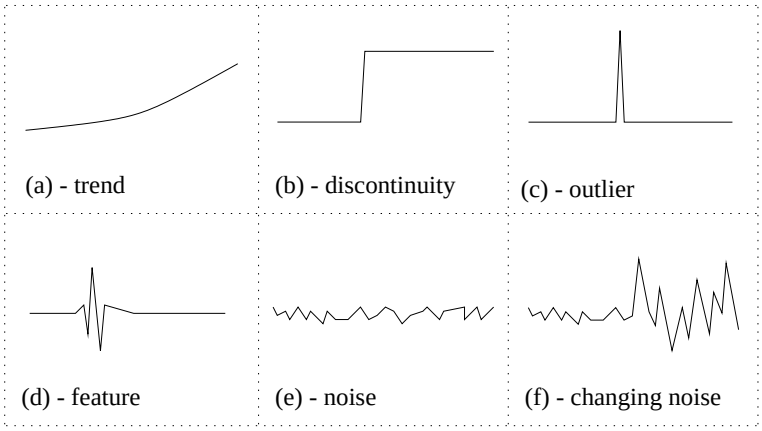


Figure 2.19: Typical phenomena in time series.

Thus, the idea of multiscale analysis is to examine the signal at various degrees of zooming or smoothing. Instead of extracting the important features from a single smoothed curve, the *comparison of representations at different scales* is used to decide about the significance of the features. The most important features are those that can be observed over a large variety of scales, whereas features that can be observed in a very limited range of scales are probably noise. An empirical motivation for this has been given by Witkin [Witkin, 1983], who observed that those features "that survive over a broad range of scales tend to leap out at the eye, while the most ephemeral are not perceived at all". But it can also be motivated by robustness arguments, since we are not interested in a feature extraction that is highly sensitive to a correctly chosen smoothing parameter. Moreover, experiments have confirmed that multiscale transformed information appears in the visual cortex of mammals [Wang and Lee, 1998].

2.2.3.1 Scale-Space Image

In our case, features are maximal segments of identical qualitative behaviour, which are localized by zero-crossings in the first and/or second derivative. For a multiscale analysis, starting from the original signal we determine the location of all zero-crossings, and continue to do so while slowly increasing the scale and therefore obtaining smoother representations of the signal. The time/scale plane is denoted as the *scale-space*, the function $F(x, s)$ yielding the value of the original function f after smoothing with a filter of size s, is called *scale-space image of* f [Witkin, 1983].[4] Figure 2.20 shows an example of an iterative application of a discrete scale-space kernel to the ATTAS signal in figure 2.1. Obviously, some features (extrema) of the series vanish more quickly than others, noise disappears more quickly than clear features.

Figure 2.21 shows the position of extrema in the scale space image, which show up as vertically oriented lines. The lines indicate the development of the extrema versus

[4] Originally, the scale s in [Witkin, 1983] was the variance σ of the Gaussian kernel, however, since we deal with the discrete case only and not necessarily with the Gaussian, $2s + 1$ denotes the number of points used to calculate the smoothed value at scale s.

scale, the longer the lines the more persistent is the extremum against smoothing. In section 2.2.2 we have discussed the discrete scale-space kernel property, which guarantees that no new extrema are introduced at coarser scales. As s decreases, additional extrema may appear, but existing ones cannot disappear. This means for the contour plot in figure 2.21 that all lines can be traced from the top (coarse scale) to the original time series (zero scale). In scale-space theory two reasonable assumptions [Witkin, 1983] are made: We assume that zero-crossings at different scales that lie on the same contour line stem from the same underlying event, which is called the identity assumption. Secondly, we assume that the true location of an event that gave rise to a zero-contour is the contour's location as $s \to 0$, which is referred to as the localization assumption.

The localization assumption allows us to compensate the temporal distortion (dislocation) of the zero-crossings at coarser scales, which was outlined as a drawback when using kernel smoothing at a single scale in section 2.2.2. Thus, we may use a coarser representation to identify important features, and use a coarse-to-fine tracking to localize the features exactly in time.

2.2.3.2 Scale-Space Primal Sketch

Since we cannot have two minima without a maximum in between, smoothing removes extrema pairwise, as can be observed in figure 2.21 and schematically in figure 2.22. Consequently, starting in the scale-space from the coarsest scale, pairs of extrema of opposite sign appear in the smoothed signal as the scale is decreased (difficult to observe in figure 2.21 for fine scales, but clearly visible for larger scales). Since a discrete scale-space kernel has been used for the construction of the contours of the zero-crossings, only at these points an interval is divided into three subintervals. Again, for each of these subintervals, either it persists down to the finest scale or is again subdivided due to the appearance of another minimum/maximum pair at a finer scale. Thus, the zero-crossings can be organized as a ternary-branching tree (with the root node at the coarsest scale).

The intervals of increasing/decreasing segments are given by the zero-crossings of the first derivative or extrema in the original function. While the location of these intervals varies with scale, coarse-to-fine tracking allows to recover the position of the associated events in the original signal. When propagating the locations from the zero scale up to all higher scales, the contour curves (figure 2.21) turn into straight vertical lines. Then, the ternary-branching tree corresponds to a hierarchical segmentation of the time axis.

Each node in the tree defines an interval in time and scale, corresponding to a rectangle in scale-space. The time interval denotes the location of the feature and the scale interval denotes the range of scales over which the feature can be observed. Collectively, these rectangles tessellate the scale-space plane. This tree is called the *interval tree of scales* or *scale-space primal sketch* of the signal [Witkin, 1983, Bakshi and Stephanopoulos, 1995]. Figure 2.23 shows the interval tree of scales for the time series in figure 2.20. It can be considered as a concise description of the qualitative development of the signal against variations in scale. The "+" and "−" signs in the rectangles indicate whether it represents an increasing or decreasing segment.

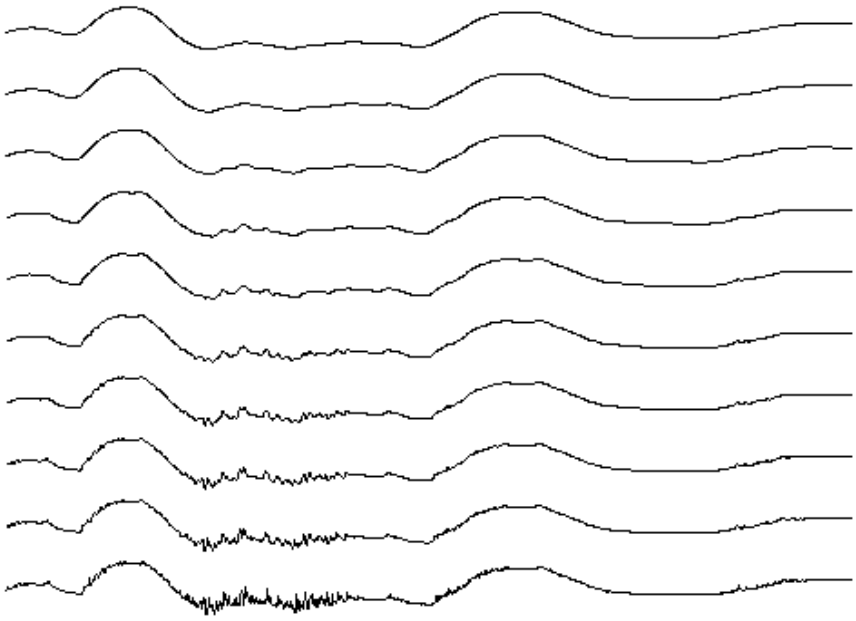


Figure 2.20: The filter $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ is applied to the original time series (bottom) and iteratively to the smoothed series for 1, 2, 4, 8, 16, ... times (from bottom to top).

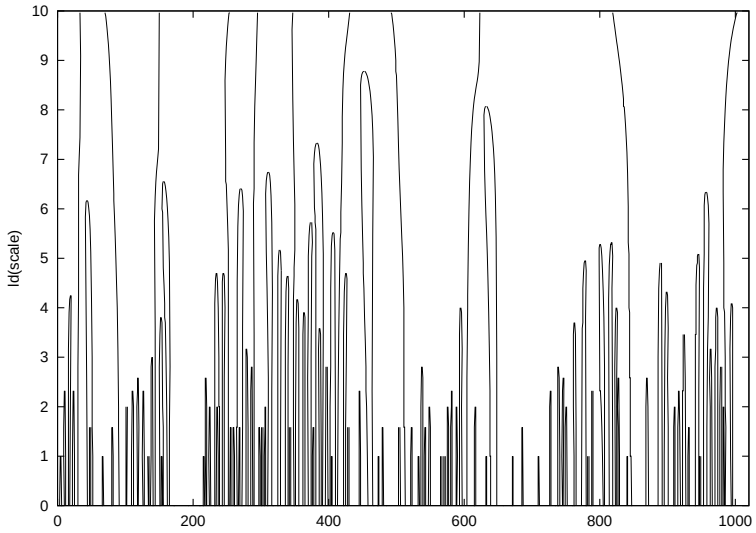


Figure 2.21: Location of local extrema versus scale (same time series as in figure 2.20).

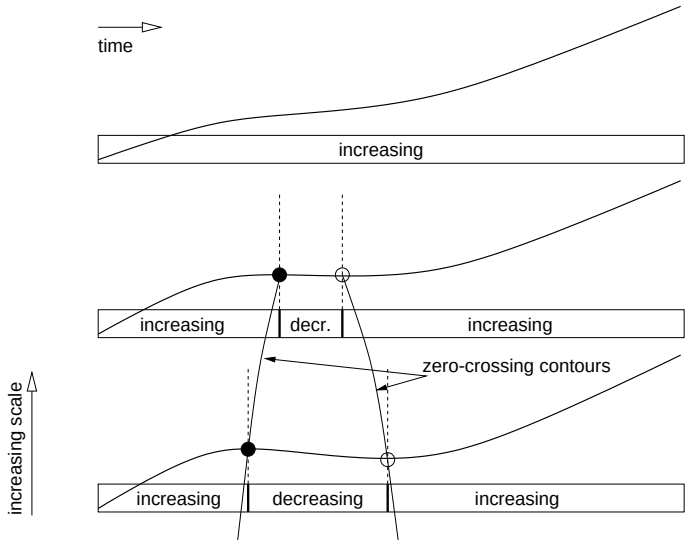


Figure 2.22: The figure shows a signal at three different levels of smoothing. At the coarsest scale, the local extrema have disappeared. As scale is decreased, extrema of opposite sign appear pairwise. At such scales, a single interval is subdivided into three subintervals.

2.2.3.3 Extraction of Stable Features

So far we have compensated the effects of dislocation, but the scale parameter is still a free variable – for every different scale we obtain a different representation of the smoothed signal. We can imagine the selected scale parameter as a horizontal line that can be shifted along the scale axis to observe different qualitative descriptions of the same signal. If this line crosses rectangles that are long-stretched in the scale axis this means that the underlying feature is very stable, since it is resistant against variations of scale. Since we are interested in the robustness of the extracted features, we want to use the scale-space lifetime over which a feature persists as a measure of robustness or significance.

For a large variety of continuous signals, the number of local extrema decreases with scale approximately as s^{α} [Lindeberg, 1993b]. A measure of scale-space lifetime should neither overestimate the lifetime of features in coarser nor in finer scales, it is reasonable to require that the expected remaining lifetime of a local extremum should not vary with scale. For the continuous case, this leads to $\log(s_D) - \log(s_A)$ as the definition for the *stability* or scale-space lifetime of an interval, where s_A denotes the scale where the feature appears and s_A where it disappears. (We have already used logarithmic scale in figure 2.23). However, for discrete signals the $s^{-\alpha}$ behaviour cannot hold, since it is based on the assumption that the original signal contains equal amounts of structure at all scales. But the discrete signal is limited by its finite sampling density. Lindeberg therefore proposes an *effective scale* τ_{eff} and defines the scale-space lifetime as $\tau_{\mathit{eff}}(s_D) - \tau_{\mathit{eff}}(s_A)$ [Lindeberg, 1993b, Lindeberg, 1993a]. Effective scale is estimated from the average density of local extrema in a set of reference signals. This notion of effective scale has been used to develop the interval tree of scales in figure 2.25(a).

Once we have defined a numerical measure for segment stability, we can seek for a single scale which maximizes the mean stability (maximum stability line in figures

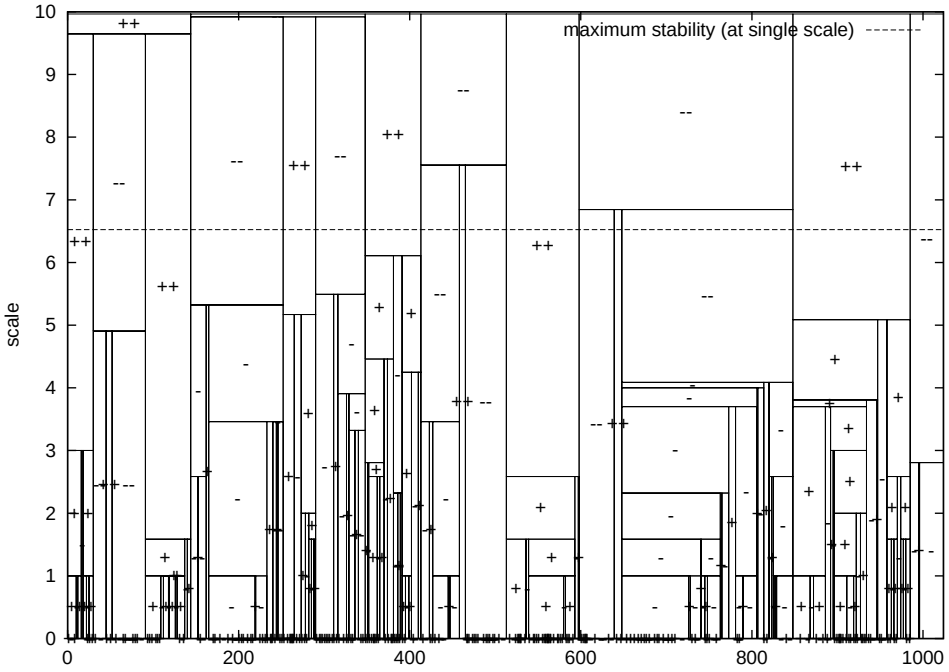


Figure 2.23: Scale-space primal sketch for figures 2.20 and 2.21. (Plus and minus symbols indicate increasing and decreasing segments.)

2.23 or 2.25(a))[5]. However, once we have done a multiscale analysis we do not have to restrict ourselves to features that appear in a single scale. Instead, Witkin proposes to descend the tree from the top to the bottom, as long as the mean lifetime of the offsprings is larger than the lifetime of any of the parents [Witkin, 1983]. The latter criterion gives signal descriptions that correspond very well to the human perception of the time series. The stable features according to this criterion are indicated by double signs in the figures ("++" and "−−").

2.2.3.4 Efficient Extraction of Stable Descriptions

Although kernel smoothing is an $O(n)$ operation, we have to apply it to a large variety of scales which makes the construction of the interval tree of scales computationally expensive. The most prominent technique for fast multiscale analysis is the wavelet analysis, which can be tailored to fit our purposes.

Robustness is a desired property of any feature extraction algorithm, that is, similar signals (say translated signals) shall lead to similar extracted features. As already mentioned, the fast discrete wavelet transform is $O(n)$ due to a dyadic tessellation of frequency and time. Due to this time tessellation, the wavelet representation is not translationally invariant. This means that a copy of a pattern, dilated by some Δt which is not a power of 2, does not manifest in identical wavelet coefficients for both patterns. This may perturb the robustness of pattern recognition algorithms quite easily. Therefore, although dyadic sampling gives the most efficient algorithm, we use uniform time sampling instead, leading to an $O(n \log n)$ algorithm. Uniform sampling is not translationally invariant itself if we translate

[5]To avoid representations that are too coarse, the best results were obtained by using the first local maximum in stability when sweeping from lower to higher scales.

by $\Delta t \notin \mathbb{Z}$, but the restriction $\Delta t \in \mathbb{Z}$ is obviously less severe. However, it has been observed in [Mallat and Zhong, 1992] that the zero crossings and extrema of a wavelet transform are translationally invariant.

Traditionally derivatives of Gaussians are used for edge detection via signal filtering [Marr and Hildreth, 1980]. The Gaussian has several unique features (cf. section 2.2.2), but is not compactly supported making the convolution with the signal expensive. Several authors suggest to use a cubic spline instead of the Gaussian [Holschneider et al., 1989, Shensa, 1992, Mallat and Zhong, 1992, Wang and Lee, 1998]. Its compact support and good correspondence to the Gaussian (cf. figure 2.24) make it a promising candidate. The resulting wavelet transform that uses the first derivative of the cubic B-spline is known as the "algorithme à trous". With this wavelet we obtain a (multiscale) estimation of the first derivative from the wavelet coefficients, which makes it particularly easy to find extrema and inflection points in the signal (zero-crossings and extrema in the first derivative, resp.). As a discrete filter, the cubic B-spline still represents a discrete scale-space kernel [Wang and Lee, 1998], which justifies the replacement of the Gaussian.

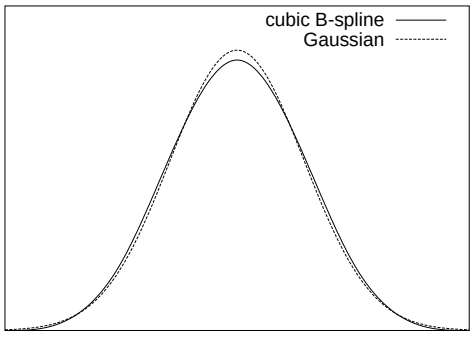
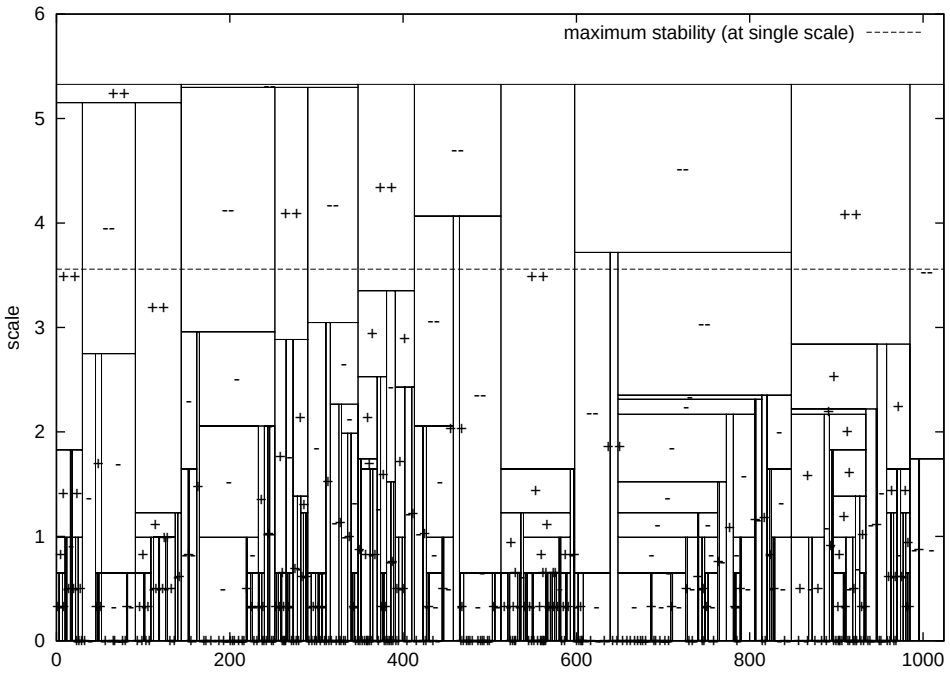


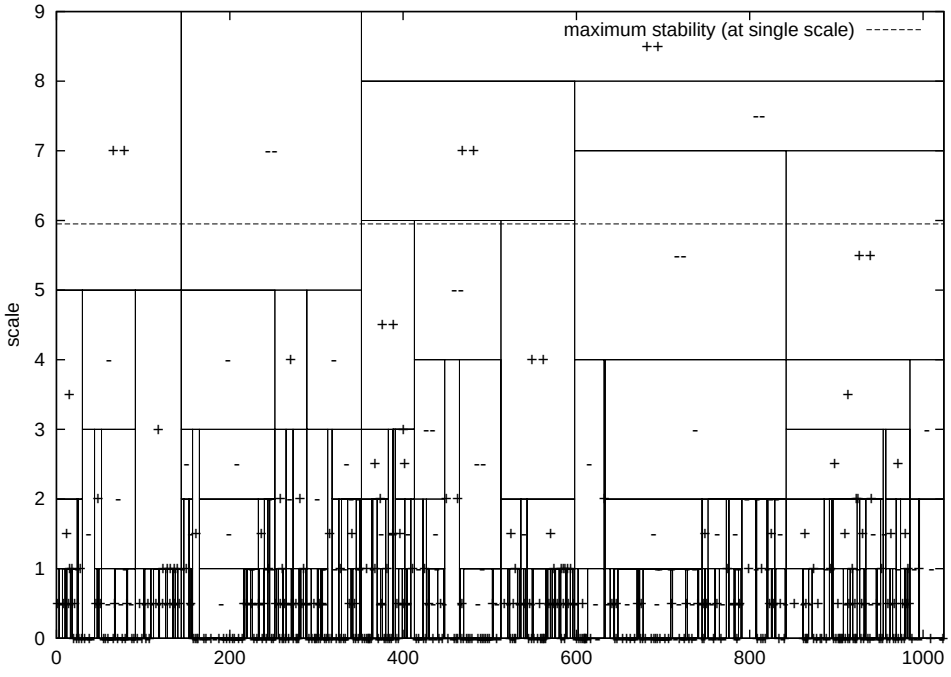
Figure 2.24: Similarity of Gaussian and cubic B-Spline.

Thus, the algorithm à trous is well suited to replace the kernel smoothing. The same procedure as before (determination of zero-crossings, localization on finest scale, extraction of stable trends) can be done with the wavelet multiscale representation, too. In [Bakshi and Stephanopoulos, 1995] the resulting tree of scales is given a different name, *wavelet tree of scales*, which emphasizes that the trees obtained by kernel smoothing and wavelet analysis are similar, but generally different. Naturally, the interval tree is much more detailed due to its finer resolution on the scale axis. While it can be shown that the à trous algorithm yields a constant signal on the coarsest scale (here $s = 10$), the smoothed signal is usually not constant when the kernel smoothing is applied 2^{10} times (compare figure 2.23 and 2.25(b)).

In figure 2.25 we compare the interval and wavelet tree of scales when using effective scale (cf. section 2.2.3.3). Then the qualitative descriptions along the scale $s \approx 3.5$, which is the most stable single scale in the interval tree of scales, matches the wavelet tree at that scale very well. It seems that the logarithmic scale of the wavelet tree corresponds very well to the effective scale in the interval tree of scale. (In figure 2.25 the iterative smoothing has been aborted after 1000 iterations, which is why we obtain a different single stable scale for the wavelet tree.)



(a) Interval Tree of Scales (filter $\frac{1}{12}(1\ 3\ 4\ 3\ 1)$)



(b) Wavelet Tree of Scales (algorithm à trous)

Figure 2.25: Wavelet and Interval Tree of Scale match pretty well if effective scale is used for the latter.

2.2.3.5 Feature Extraction from Multiscale Analysis

As already indicated before, the labels used to characterize the time series locally may be attributed appropriately, for instance, a "linearly increasing" label may be attributed with the slope. Later in the knowledge discovery process (see section 4.2) it may turn out that the slope is an informative attribute that can be used to increase the confidence in the rules we have found. The slope may be obtained from a least-squares fit of a line to the respective segment.

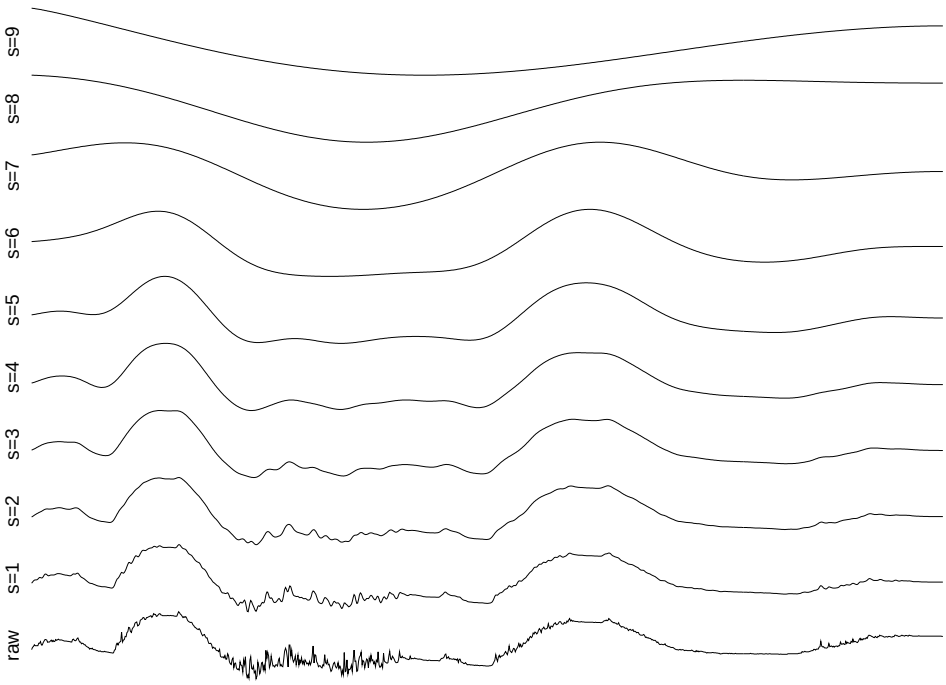
A problem that is inherent with all multiscale methods is that of consistent attributes over varying scales. We have already discussed in section 2.2.2 that smoothing dislocates features and generally "flattens" the signal (cf. figure 2.18). In principle, the wavelet technique also performs smoothing, but rather than using a single kernel a variety of smoothing kernels are applied. If we smooth the same signal by a very small (fine scale) and a very broad (coarse scale) low-pass filter and compare the slopes of the same linear segment we will obtain different values (in general, the absolute value of the slope of the latter signal will be smaller). This has considerable impact on pattern identification, because the extracted features do not only depend on the signal itself but also on the scale from which they have been extracted.

Figure 2.26 illustrates this with an example. Starting from a fully recovered signal (raw data) we start to set wavelet coefficients to zero scale by scale. Consider the case of $s = 4$: In the qualitative description of the signal at that scale the increasing and decreasing segments denoting the small peak right before the second "hill" are still present – but in the smoothed curve this hill has *almost* disappeared.

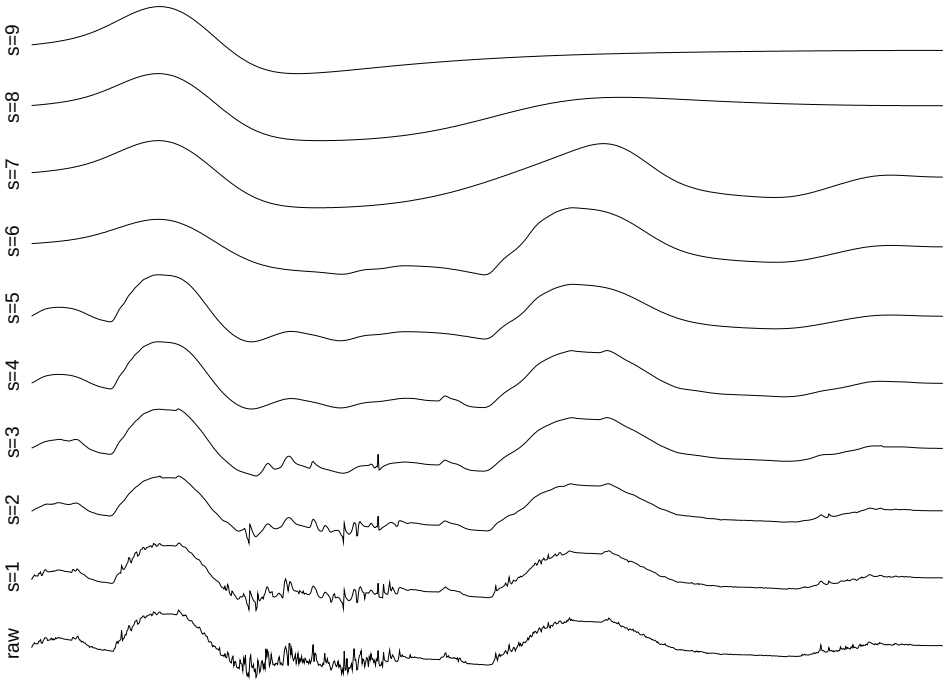
Rather than setting all coefficients in finer scales $s < 4$ to zero, we can keep those coefficients that contribute to this peak in the finer scales without changing the qualitative description. The wavelet tree does not only tessellate the scale/time plane but also the wavelet coefficients. Removing features from the signal corresponds to removing rectangles from the wavelet tree and to setting the associated wavelet coefficients to zero. Thereby we remove much fewer coefficients than before and better preserve the shape of the original signal. Selecting coefficients according to the tree of scales guarantees that we do not introduce noise or other features again, the reconstructed signal corresponds to the selected qualitative description [Bakshi and Stephanopoulos, 1995]. Figure 2.26(b) shows the resulting signals obtained by this technique, the addressed peak is much closer to the original signal while it is qualitatively unchanged. Thus, the tree of scales does not only help in recovering the dislocation but also the flattening effects of smoothing.

2.3 Summary

The methods in sections 2.1 and 2.2.1-2.2.2 are single-scale methods, because the signals are considered in a single representation. Among these methods there is no single best since their performance strongly depends on the correct parameter setting. All of them require either the specification of the number of knots k, maximal error ε, filter coefficients or something related. Such parameters can be determined using a small number of trials given a reference signal, but this is only reliable if it is reasonable to assume that unseen data will be similar in structure, roughness, and noise level. In general it will be difficult to guarantee this in advance for unseen data, especially in a data mining environment, where the measurements may be collected over years and equipment degradation may cause an increase in noise, whereas equipment renewal or replacement may cause a decrease.



(a) Wavelet coefficients are set to zero scale by scale.



(b) Wavelet coefficients are set to zero according to the wavelet tree of scales.

Figure 2.26: Reconstruction of the signal from a reduced number of wavelet coefficients. Qualitatively (in terms of increasing and decreasing segments) both reconstructions are identical. Quantitatively, however, the second reconstruction is much closer to the original signal.

In contrast, the multiscale techniques from section 2.2.3 do not make such restrictive assumptions on the (number or kind of) basis functions, nor require the a priori determination of any thresholds. They can be implemented efficiently by means of wavelets. Although a heuristic may be used (e.g. Witkins stability criterion) to select the final qualitative representation, it is well-motivated by robustness requirements and adapts to the peculiarities of the present signal, whereas other methods often use heuristics to fix the required thresholds once and then leave them unchanged during the abstraction.

In summary, we have discussed methods and means to extract labeled intervals from time series that characterize the behaviour or local trend of the series. We believe that (a) the use of higher-representations is very important if we want a human to *understand* the discovered relationships and think that this is a prerequisite for *knowledge* discovery. Since humans are used to work with graphical representations of time series, abstracted representations are necessary to account for the human way of perceiving time series. And from the reviewed abstraction methods we believe that (b) most of these methods are not very well suited for KDD and data mining, where assumptions about constant noise etc. are not valid. It is important to have a method that provides a robust and valid abstraction: With multiscale methods robustness can be obtained by considering only features that persist over a broad range of scales, and validity of the extracted features is achieved by tracking the intervals from coarse to fine scales to compensate dislocation and flattening effects.

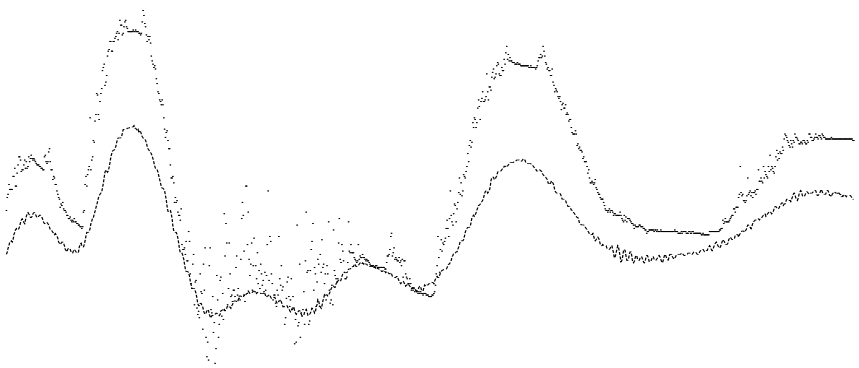


Figure 2.27: Signal of reference device.

The signal in figures 2.1, 2.10, 2.13, 2.20, etc., has been used throughout this chapter. For the sake of completeness here is some background information. It is a signal taken from a device outside the hull of the ATTAS airplane of the German Aerospace Center. The device was tested whether it is suited to substitute another (more expensive) device. Figure 2.27 shows both, the analyzed signal with dots and the signal of the reference device with a solid line. Obviously, the new device is extremely sensitive and occasionally produces a large amount of noise. The qualitative description of the reference signal is close to what has been extracted from the noisy signal from the interval tree of scales at the maximally stable scale in figure 2.23 or 2.25(a).

Chapter 3

Discovery of Temporal Patterns

Having discussed the segmentation of time series into meaningful intervals in the previous chapter, now we are interested in finding temporal patterns in a sequence of labeled intervals. Compared to other kinds of data, interval data is seldomly used in knowledge discovery. This might be related to the fact that most of the data analysis methods assume static data, that is, they do not consider time explicitly. Usually, the value of attributes is provided for a single point in time, like "patient A has disease B". If we observe the attributes over a period of time, we have to attach a time interval in which the attribute holds, for example "patient A has had disease B from 1^{st} to 7^{th} of July". It may happen that patient A gets disease B a second time, therefore sequences of labeled intervals can be viewed as a natural generalization of static attributes to time-varying domains. Again, compared to static data analysis, much less work has been done to analyse temporal data. In this chapter, we discuss a novel approach to the analysis of interval data. After providing a definition of a pattern space for interval data and a solution to the problem of measuring the frequency of occurrences of such patterns, we concentrate on efficient algorithms to discover all patterns in a given interval sequence and to estimate their frequency for further analysis in the consecutive chapters.

Let $\mathcal{S}$ denote the set of all possible trends, properties, or states that we want to distinguish, for example "pressure goes down" or "water level is constant". Such a description $s \in \mathcal{S}$ holds during a period of time $[b, f]$ where b and f denote the *initial point* in time when the description holds and the *final point* in time when the description is no longer valid. We will consider s as a descriptive label of the interval $[b, f]$. Since s describes a variable over a period of time, we may think of a system being in a certain *state* during that interval.

Definition 1 (Labeled Interval Sequence) *Given a sequence of non-empty intervals $[b_i, f_i]$ where each interval carries a label or symbol $s_i \in \mathcal{S}$. We say*

$$(b_1, f_1, s_1), (b_2, f_2, s_2), (b_3, f_3, s_3), (b_4, f_4, s_4), ...$$

is a labeled interval sequence over $\mathcal{S}$, if

$$\forall (b_i, f_i, s_i), (b_j, f_j, s_j), i < j : \qquad f_i \geq b_j \Rightarrow s_i \neq s_j \tag{3.1}$$

holds.

The definition does not require that the intervals are disjoint, which enables us to mix up several labeled interval sequences (possibly obtained from abstracting different time series) into a single one. Thus, there is no need to distinguish between uni- and multivariate analysis in the case of labeled interval sequences.[1] However, the definition does require that every triple (b_i, f_i, s) is *maximal* in the sense, that there is no (b_j, f_j, s) in the series such that $[b_i, f_i]$ and $[b_j, f_j]$ overlap or meet each other. If (3.1) is violated, we can merge both state intervals and replace them by their union $(\min(b_i, b_j), \max(f_i, f_j), s)$. We will refer to (3.1) as the *maximality assumption*.

As an example, we could have classified the points in a time series into qualitative states "increasing", "decreasing", and "constant". These three states partition the time series completely, that is, any state is continued without gap by another state (intervals will meet). But it is also possible to use only primitive patterns like "increasing" and "highly increasing". The missing patterns (e.g. "decreasing") will cause some gaps in the description of a particular time series, but this does not hinder the analysis of the sequence. In what follows, we assume that such a state sequence is given or has been derived using methods from chapter 2.

The remainder of this chapter is organized as follows: In section 3.1 we define temporal patterns formally, before we discuss means how to rate their frequency of occurrence in section 3.2. To enumerate all frequent patterns efficiently we will adapt techniques from association rule mining [Agrawal et al., 1996, Mannila et al., 1997] in section 3.3. Some experiments are conducted in section 3.4. From the frequent patterns we will then create temporal rules in the next chapter.

3.1 Definition of Temporal Patterns

In chapter 2 we have discussed how to extract temporal attributes from time series. A temporal attribute characterizes the time series locally, namely in the specified interval. More complicated descriptions can be obtained by using multiple temporal attributes and their pairwise relationships. For instance "water level constant" while "valve is closed" followed by "water level increases" while "valve is open". Or "turning keys" starts "motor is running" and during "motor is running" we have "pushing gas pedal" starts "speed is increasing". We want to derive such patterns from observations in the sequence.

To define a more abstract notion of a temporal pattern composed out of temporal attributes we use Allen's temporal interval logic [Allen, 1983] to describe the relation between the time intervals. For any pair of intervals we have 13 possible relationships; they are illustrated in figure 3.1. For example, we say "A meets B" if interval A terminates at the same point in time at which B starts. The inverse relationship is "B is-met-by A". In the following we will abbreviate the interval relations as shown in the figure. The set of all interval relationships is denoted by $\mathcal{I}$. For two intervals I_1 and I_2 we denote their temporal relationship by $\mathrm{ir}(I_1, I_2) \in \mathcal{I}$.

Given n state intervals (b_i, f_i, s_i), $1 \leq i \leq n$, we can capture their relative positions to each other by an $n \times n$ matrix R whose elements $R[i, j] \in \mathcal{I}$ describe the relationship between state interval no. i and j. As an example, let us consider the sequence in figure 3.2. Obviously state A is always followed by B. And the gap between A and B is covered by state C. Below the state interval sequence both of these patterns are written as a matrix of interval relations.

[1] Note that most of the traditional methods for time series analysis cover the uni- and bivariate case only.

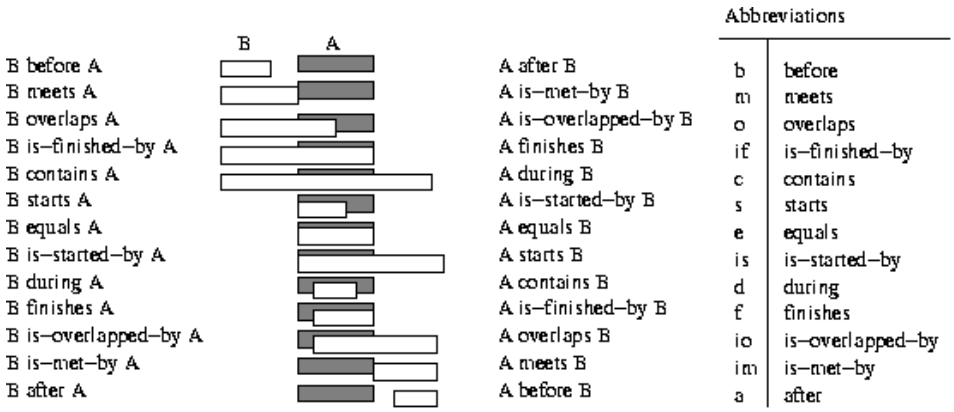


Figure 3.1: Allen's interval relationships.

Definition 2 (Temporal Pattern) *A pair* $P := (s, R)$*, where* $s : \{1, .., n\} \to \mathcal{S}$ *and* $R \in \mathcal{I}^{n \times n}$*,* $n \in \mathbb{N}$*, is called a "temporal pattern of size* n*" if there is a state sequence* $(b_i, f_i, s_i)_{1 \leq i \leq n}$ *such that* $s(i) = s_i$ *and* $R[i, j] = ir([b_i, f_i], [b_j, f_j])$*. By* $\dim(P)$ *we denote the dimension (number* n *of intervals) of the pattern* P*. If* $\dim(P) = n$*, we say that* P *is a* n*-pattern.*

Of course, many different subsequences map to the same temporal pattern. We call these sequences *instances* of the temporal pattern. By $TP(\mathcal{S})$ we denote the space of all temporal patterns of arbitrary dimension using symbols from $\mathcal{S}$.

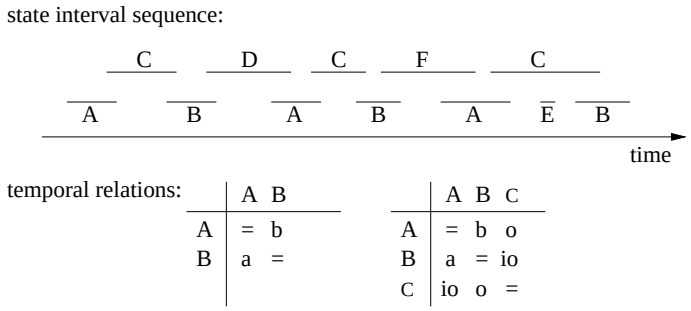


Figure 3.2: Example for state interval patterns expressed as temporal relationships.

Note that we allow for atomic relations in $R[i, j]$ only. In the full-fledged interval algebra disjunctive composition is possible, for instance "(A *before* B) $\vee$ (A *after* B)"="A $\langle$*before, after*$\rangle$ B" (that is, A and B are disjoint). Most (traditional) applications of Allen's interval algebra start with a description of the temporal pattern (allowing disjunctive combinations) and it is an NP-complete problem to decide whether there exists an instance of this pattern. As an example, consider constraints on the various jobs in a manufacturing task, then an instance of the pattern is a job schedule (the problem of finding such an instance is called *constraint satisfaction problem* (CSP)). Here we have the reverse case: we start with a realisation (the given sequence) and are interested in a higher-level description. We will return to the disjunctive combination of atomic interval relationships later in section 4.1.4 and chapter 6.

3.1.1 Rules induced by Subpatterns

Next, we define a partial order $\sqsubseteq$ on temporal relations. Informally, from a temporal pattern B we obtain a pattern $A \sqsubseteq B$ by removing any number of states in an instance of A.

Definition 3 (Subpattern Relation) *We say that the temporal pattern (s_A, R_A) is a subpattern of (s_B, R_B) (or $(s_A, R_A) \sqsubseteq (s_B, R_B)$), if $\dim(s_A, R_A) \leq \dim(s_B, R_B)$ and there is an embedding (injective mapping) $\pi : \{1, .., \dim(s_A, R_A)\} \to \{1, .., \dim(s_B, R_B)\}$ such that*

$$\begin{gathered} \forall i, j \in \{1, .., \dim(s_A, R_A)\} : \\ s_A(i) = s_B(\pi(i)) \quad \wedge \quad R_A[i, j] = R_B[\pi(i), \pi(j)] \end{gathered} \tag{3.2}$$

The relation $\sqsubseteq$ is reflexive and transitive, but not antisymmetric: we can have $(s_A, R_A) \sqsubseteq (s_B, R_B)$ and $(s_B, R_B) \sqsubseteq (s_A, R_A)$ without $s_A = s_B$ and $R_A = R_B$ due to a different state ordering. But permutating the intervals (including the associated rows and columns) does not change the semantics of the temporal pattern (a pictorial representation would be identical). Therefore, we define $(s_A, R_A) \equiv (s_B, R_B) :\Leftrightarrow (s_A, R_A) \sqsubseteq (s_B, R_B) \wedge (s_B, R_B) \sqsubseteq (s_A, R_A)$ and consider the factorisation $({}^{TP(\mathcal{S})}/_{\equiv}, {}^{\sqsubseteq}/_{\equiv})$, where $\sqsubseteq$ has been generalized canonically to equivalence classes. Then, ${}^{\sqsubseteq}/_{\equiv}$ is also antisymmetric and thus a partial order on (equivalence classes of) temporal patterns.

Given two temporal patterns $X \sqsubseteq Y$ we can formulate rules

$$\text{if } X \text{ then } Y \text{ with probability } p \tag{3.3}$$

modelling temporal dependencies between states in X and Y. We call the pattern X in the premise the *premise pattern* and the pattern Y in the conclusion the *rule pattern*. The rule pattern also comprises the premise pattern ($X \sqsubseteq Y$). If we remove the premise from the rule pattern we obtain the *conclusion pattern*. Note that it is not possible to reconstruct the rule pattern from premise and conclusion pattern only, since the rule pattern additionally determines the temporal relationship between two intervals where one of them stems from the conclusion and the other from the premise. We have to postpone how to obtain the rule probability p until section 3.2.

3.1.2 Normalized Form of a Temporal Pattern

To simplify notation we pick a subset $NTP(\mathcal{S}) \subset TP(\mathcal{S})$ of *normalized* temporal patterns such that $NTP(\mathcal{S})$ contains one temporal pattern for each equivalence class of ${}^{TP(\mathcal{S})}/_{\equiv}$ and $(NTP(\mathcal{S}), \sqsubseteq)$ is isomorphic to $({}^{TP(\mathcal{S})}/_{\equiv}, {}^{\sqsubseteq}/_{\equiv})$. In the remainder, we will then use $(NTP(\mathcal{S}), \sqsubseteq)$ synonymously to $({}^{TP(\mathcal{S})}/_{\equiv}, {}^{\sqsubseteq}/_{\equiv})$.

The intuitive idea is to order the state intervals in time with increasing index. However, this ordering is slightly more complex with arbitrary intervals than with points. Given an order on states $(\mathcal{S}, <)$, we say that a temporal pattern (s_A, R_A) *has normalized form*, if for all $1 \leq i \leq \dim(s_A, R_A)$ and $i < j \leq \dim(s_A, R_A)$ the following conditions hold:

- *Case 1:* $s(i) = s(j)$. If we have two intervals with identical states, then by the maximality assumption (3.1) there must be a time gap between the intervals, otherwise we could merge both state intervals into a single new one, which

contains both intervals. Therefore, in this case we have only two possible relations $R[i,j] \in \{before, after\}$. To preserve temporal ordering we require $R[i,j] = before$.

- *Case 2:* $s(i) \neq s(j)$.
  - *Case 2a: distinct initial times.* If the initial times of both intervals are different, we use the ordering of the initial times, that is, in a normalized form we have $R[i,j] \in \{contains, is\text{-}finished\text{-}by, overlaps, meets, before\}$ (cf. figure 3.1).
  - *Case 2b: initial times coincide.* Thus we have $R[i,j] \in \{equals, starts, is\text{-}started\text{-}by\}$. If both intervals are identical, we use the order on the states, that is, in a normalized form we require $s(i) < s(j)$ (note that we are sure that $s(i) \neq s(j)$ in this subcase). If the final times are different, we require $R[i,j] = starts$ to make sure that the interval with index i ends before the interval with index j.

This gives us the following definition of a normalized form:

Definition 4 *A temporal pattern (s,R) is said to be in normalized form, if for all i,j the following condition holds:*

$$\begin{aligned} i < j \quad \Leftrightarrow \quad & (s(i) = s(j) \wedge R[i,j] = before) \\ \vee \quad & (s(i) < s(j) \wedge R[i,j] = equals) \\ \vee \quad & (s(i) \neq s(j) \wedge \left(R[i,j] \in \begin{array}{c} \{starts, contains, is\text{-}finished\text{-}by, \\ overlaps, meets, before\} \end{array} \right) \end{aligned} \tag{3.4}$$

While by definition a temporal pattern has no specific temporal extension (because it has been abstracted from the time intervals given in a specific sequence), the pattern instances themselves do have a temporal extension. If the sequence is given, we achieve a normalized pattern simply by sorting the state intervals lexicographically:

Theorem 1 *Let P denote a normalized temporal pattern obtained from a sequence of length L and let (b_i, f_i, s_i) denote the respective state that is addressed by $s_P(i)$. For any $i,j \in \{1,...,L\}$ we have*

$$i < j \Rightarrow (b_i, f_i) \leq (b_j, f_j) \tag{3.5}$$

and

$$i < j \Leftrightarrow (b_i, f_i, s_i) < (b_j, f_j, s_j), \tag{3.6}$$

where lexicographical ordering is used for tuple comparisons.

Proof of Theorem 1: (3.5): $\Rightarrow$: Let $i < j$. According to (3.4) we have $R[i,j] \in \{starts, equals, contains, is\text{-}finished\text{-}by, overlaps, meets, before\}$. For $R[i,j] = equals$ we have $b_i = b_j$ and $f_i = f_j$ and therefore $(b_i, f_i) = (b_j, f_j)$. For $R[i,j] = starts$ we have $b_i = b_j$ and $f_i < f_j$ and therefore $(b_i, f_i) < (b_j, f_j)$. In all other cases we have $b_i < b_j$ and therefore $(b_i, f_i) < (b_j, f_j)$.

(3.6): $\Rightarrow$: Continuing from the last paragraph, there was only a single case of equality (in all other cases, the lexicographical $<$ relation has been determined on (b_i, f_i), it will therefore also hold on (b_i, f_i, s_i)). Again, according to (3.4) we know $s_i = s_P(i) < s_P(j) = s_j$ and therefore $(b_i, f_i, s_i) < (b_j, f_j, s_j)$. $\Leftarrow$:

Let $(b_i, f_i, s_i) < (b_j, f_j, s_j)$. *(a) Order is decided on* $b_i < b_j$. Then we have $R[i,j] \in \{\textit{contains}, \textit{is-finished-by}, \textit{overlaps}, \textit{meets}, \textit{before}\}$. In case of *before* it follows immediately from (3.4) that $i < j$. For all other cases we have to show $s_i \neq s_j$ first. For all possible relationships besides *before* we can conclude that $[b_i, f_i] \cap [b_j, f_j] \neq \emptyset$, by the maximality assumption (3.1) we thus obtain $s_i \neq s_j$. *(b) Order is decided on* $f_i < f_j$, that is, $b_i = b_j$. Then we have $R[i,j] = \textit{starts}$ and again $i < j$ by (3.4) ($s_i \neq s_j$ as in case (b)). *(c) Order is decided on* $s_i < s_j$, that is, $b_i = b_j$ and $f_i = f_j$. Then, $R[i,j] = \textit{equals}$ and due to (3.4) we have $i < j$. ■

Thus, every temporal pattern has a unique normalized form (obtained by sorting the intervals of a pattern instance lexicographically). Having uniqueness, we can check for equivalence of temporal patterns (s_A, R_A) and (s_B, R_B) in their normalized form simply by checking $s_A = s_B \wedge R_A = R_B$. Every equivalence class of temporal patterns contains a unique normalized pattern, which is why we restrict ourselves to normalized patterns in the remainder.

3.2 Occurrences of Temporal Patterns in Interval Sequences

Having defined temporal patterns, we are now interested in means to rate them with respect to the frequency of their occurrences: The *support* of a pattern denotes how often a pattern occurs. What is a suitable definition of support in the context of temporal patterns? Perhaps the most intuitive definition is the following: The support of a temporal pattern is the number of instances of the temporal pattern in the sequence. Let us examine this definition in the context of the following example (see figure 3.1 for abbreviated interval relationships; the temporal pattern is depicted below the relation matrix):

if

	A	B
A	$=$	b
B	a	$=$

A B

then

	A	B	A	B
A	$=$	b	b	b
B	a	$=$	b	b
A	a	a	$=$	m
B	a	a	im	$=$

A B A B

with probability p (3.7)

How often does the pattern in the conclusion occur in the sequence in figure 3.3(a)? We can easily find 3 occurrences as shown in figure 3.3(b). The remaining (unused) states do not form a fourth pattern. How often does the premise pattern occur? By pairing states (1,4), (2,6), (3,7), etc. we obtain a total number of 7. So we have $p = \frac{3}{7}$ because 3 out of 7 premise patterns can be completed to rule patterns. This may correspond to our intuitive understanding of the rule, but we can improve p to $\frac{4}{7}$ when using the rule pattern assignment in figure 3.3(c). The latter assignment is perhaps less intuitive than the first, because the pattern's extension in time has increased. But now we have a sequence that is assembled completely out of rule patterns, there is no superfluous state. Then, would not it be more natural to have a rule probability near 1 instead of $\frac{4}{7}$?

The purpose of the example is to alert the reader that rule semantics is not that clear as might be expected. The latter problem regarding the temporal extension of the patterns can be accomplished by introducing a mask or window such that we count only local patterns that are sufficiently close together. We therefore choose a maximum duration Δt_{win}, which serves as the width of a sliding window which is

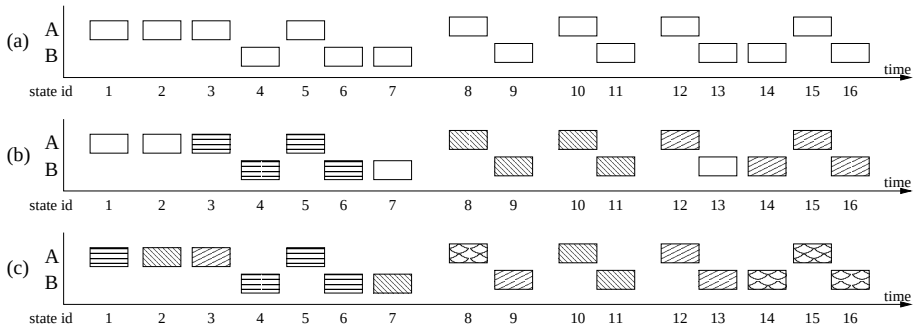


Figure 3.3: Counting the occurrences of temporal patterns. (States with different labels (A and B) are drawn on different levels. Note that the pattern of interest (3.7) requires a *meets* relation in the conclusion.)

moved along the sequence. We consider only those pattern instances that can be observed within this window. In a monitoring and control application, the threshold Δt_{win} could be taken from the maximum history length that can be displayed on the monitor and thus be inspected by the operator. This is a reasonable assumption: the whole pattern has to be small enough to be observable by a (forgetful) operator. The right bound of the sliding window serves as a reference point and will be denoted by t_{act}.

However, determining the maximum number of pattern occurrences is a complex task and does not necessarily correspond to our intuitive counting. Introducing a constraint on the pattern extension does not simplify the pattern counting, we still have to take the complete series into account in order to find the maximum number of pattern occurrences. Figure 3.4(a) illustrates this with an example, the sliding window is indicated by a shaded rectangle. Maximizing the number of patterns within the window leads us to the assignment in subfigure (a). The remaining states do not form a third occurrence. However, if we choose a different assignment at the shown position (yielding only one pattern occurrence in the first window), we obtain two additional occurrences when shifting the window along the series (subfigure (b)).

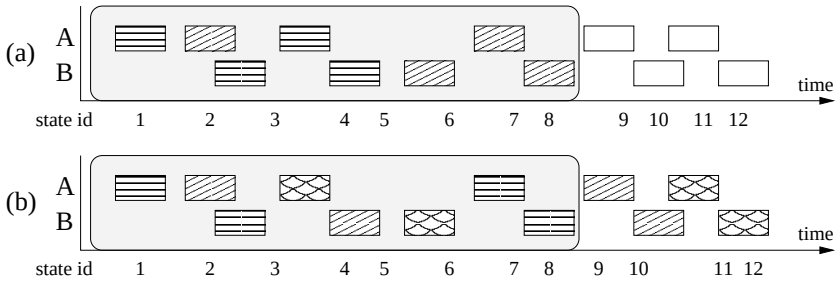


Figure 3.4: Counting the occurrences of temporal patterns within a sliding window.

If we denote the number of states in the sequence by L and the (average) number of states in the window by R, the example has shown that the complexity of pattern counting is still $O(c(L))$ and not $O(c(R))$ (where $c(n)$ denotes the complexity of the counting process in a series of n states). Due to the variability of temporal patterns c increases quickly with n. Therefore, *counting* the number of pattern occurrences becomes computationally intractable.

A solution to this problem is to select a different support definition:

Definition 5 (Support of Temporal Pattern) *We define the total time in which (one or more) instances of P can be observed in the sliding window as the support of the pattern:*

$$\text{supp}(P) = \int_t \chi_{O_P}(t)dt$$

where $O_P = \{t \,|\, P$ is observable in sliding window at position $t_{act} = t\}$ and χ_{O_P} is the characteristic function of O_P (that is, $\chi_s(t) = 1 \Leftrightarrow t \in O_P$ and 0 otherwise).

We will see shortly that the time points in which P is observable can be written as a sequence of intervals $O_P = \{[t_1, t_2], [t_3, t_4], ..., [t_{2n-1}, t_{2n}]\}$. Then the support is given by $\text{supp}(P) = \text{len}(O_P) := \sum_{i=1}^{n} t_{2i} - t_{2i-1}$ which accumulates the lengths of the observation periods.

Now, for each window, we have a complexity of $O(t(R))$, where $t(n)$ is the complexity of testing whether a pattern is contained in a sequence of n states. Testing for all window positions is then $O(L \cdot t(R))$ which is usually much smaller than $O(c(L))$ because enumeration of all instances is more complex than finding at least one instance and $R \ll L$.

However, changing the support definition always changes the semantics of a rule. What is the semantics of the rule probability p? Obviously we have

$$\text{supp}(P) \leq \Delta t_{win} + \underbrace{\max\{f_i \,|\, 1 \leq i \leq L\} - \min\{b_i \,|\, 1 \leq i \leq L\}}_{\text{sequence length in time}}$$

If we divide the support of a pattern by the sequence length in time plus the window width Δt_{win} we obtain the relative frequency p of the pattern: If we randomly select a window position we can observe the pattern with probability p. This is a nice interpretation, which in some cases might be superior to the pattern counting semantics. If we choose a sliding window width that is sufficiently large in figure 3.3(a), we will always observe at least one occurrence of both the premise and conclusion pattern in our exemplary rule. Thus, we obtain a rule confidence near 1, which is somewhat more desirable than $\frac{4}{7}$. (Yet, it is still not perfect, as we will discuss later in section 4.1.1.)

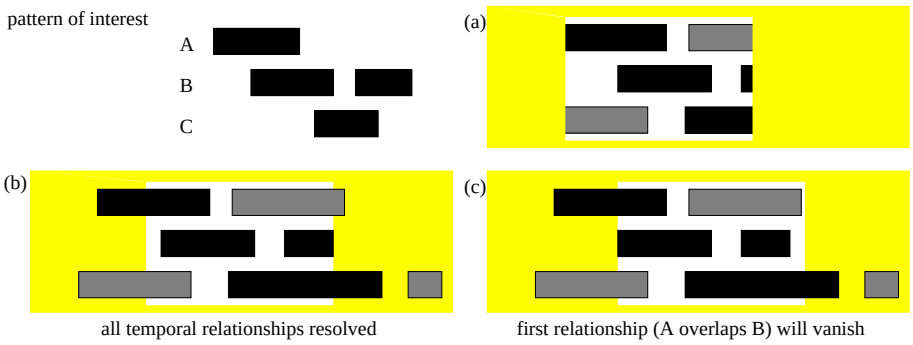


Figure 3.5: Visibility of temporal patterns. Although it seems that the pattern of interest (top right) is contained in the sliding window position (case (a)), it turns out later (bottom positions) that it is not. In position (b) the pattern "A overlaps B, B overlaps C, C contains a second B" becomes visible, in position (c) it disappears.

3.2.1 Duration of Obervation

But how do we decide if a pattern is observable within the sliding window? Consider a pattern "A overlaps B, B overlaps C, C overlaps a second B", which is also depicted in the top left corner of figure 3.5. Can we observe this pattern in the sliding window position (a) of figure 3.5? It seems so, but we cannot be quite sure, as the bottom row illustrates. Here, not only the window content but the whole sequence is visible. We can see that we have a "C contains a second B" relationship rather than an "overlaps" relationship. We start to observe the "contains"-pattern as soon as all temporal relationships can be resolved (position (b)) and stop seeing it as soon as one temporal relationship becomes unknown again (position (c)).

To further illustrate the definition of support with some examples consider figure 3.6. In subfigure (a) we have a single state A. We see the pattern for the first time, when the right bound of the sliding window touches the initial time of the state interval (dotted position of sliding window). We can observe A unless the sliding window reaches the position that is drawn with dashed lines. The total observation time is therefore the length of the sliding window Δt_{win} plus the length of state interval A. The support (observation duration) is depicted at the bottom of the subfigure. Apparently we obtain observation (or support) *intervals* whose bounds are given by the reference point t_{act} of the respective sliding window.

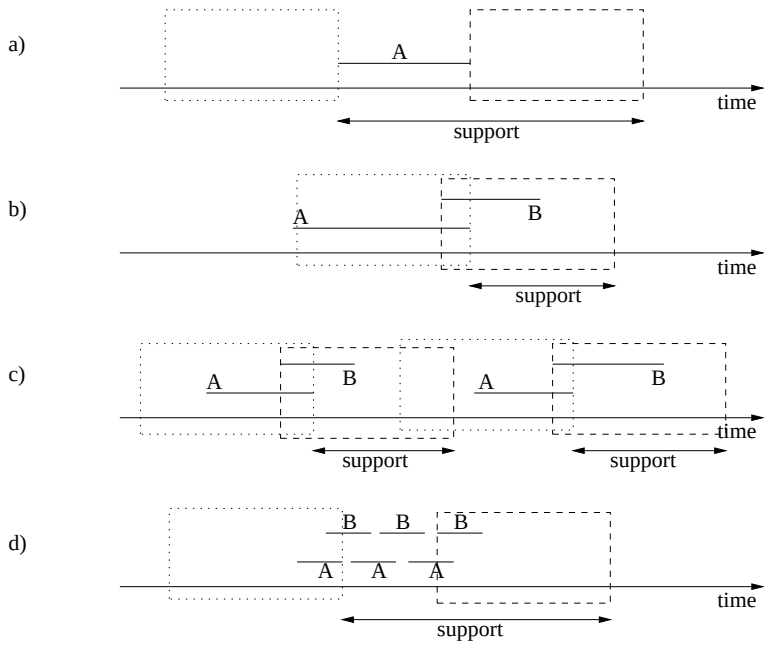


Figure 3.6: Illustration of support: Intervals are shown as lines, the rectangle denotes the sliding window.

Subfigure (b) shows another example "A overlaps B". Note that the pattern is not yet visible when the window meets B, because the temporal relationship between A and B is not yet resolved. Once A has ended (but not B) we see that the temporal relationship is *overlaps* (dotted window). We loose the instance as soon as we loose the possibility to compare the initial point of A with the inital point of B (dashed window). If the pattern occurs multiple times, two things may happen: If there is a gap between the pattern instances, such that we loose the pattern in the meanwhile, then the support of the individual instances add up to the support of the pattern,

as shown in subfigure (c). If there is no such gap (subfigure (d)), we see the pattern as soon as a first instance enters the sliding window until the last instance leaves the window. In the meantime, it does not matter *how many* instances are present, as long as there is at least one.

For a pattern P we denote the set of reference points t_{act} for which P has been observed in the window by O_P. Figure 3.6 also illustrates that the reference points that contribute to the support of a pattern themselves can be characterized as a sequence of (unlabeled) intervals. We will make use of this fact later in section 3.3.

We note in passing, that the rule probability $p = \frac{3}{7}$ in figure 3.3 is obtained by using the concept of counting minimal occurrences only. Minimal occurrences have been used in [Mannila et al., 1997] for the discovery of frequent episodes in event sequences. An instance of a pattern P in a time interval $[t_0, t_3]$ is a minimal occurrence, if there is no $[t_1, t_2] \subset [t_0, t_3]$ such that there is also an instance of P within $[t_1, t_2]$. We do not follow this idea, since we consider the rule discovery to be less robust when using minimal occurrences. Consider a pattern "A before B", where the length of the intervals is characteristic for the pattern. If the interval sequence is noisy, that is, there may be additional short B intervals in the gap of the original pattern, the minimal occurrence of A and noisy B would prevent the detection of A and original B. Rule specialization as we will discuss in chapter 4 would not have a chance to recover the original pattern. Such a situation can easily occur in an automatically generated sequence which describes the local trend of a time series, where noise in the time series will cause noisy trend segments in the labeled interval sequence (where the labels denote the local trend).

Note that our definition of support does not imply that the whole pattern instance $P := \{(b_i, f_i, s_i) \,|\, 1 \leq i \leq n\}$ must fit into the sliding window. Denoting the temporal extent (or duration) of P by

$$D(P) := \max\{f \,|\, (b, s, f) \in P\} - \min\{b \,|\, (b, s, f) \in P\},$$

we do not have $D(P) \leq \Delta t_{win}$ in general. Figure 3.7 illustrates this fact in case there are intervals with a length that exceeds Δt_{win} (window drawn with solid lines). State A lasts for a time period that is longer than Δt_{win}, nevertheless we can observe the pattern "D after C, A contains C and D" within the window. The pattern "B before C" in the window drawn with dashed lines is another example where we can observe the pattern although $D(P) > \Delta t_{win}$. However, we can not (yet) observe "A contains C" in the dashed window, because the final time of C is not yet visible – within the limited scope of the sliding window we cannot decide whether there is an an *overlaps* or *finishes* relation to come.

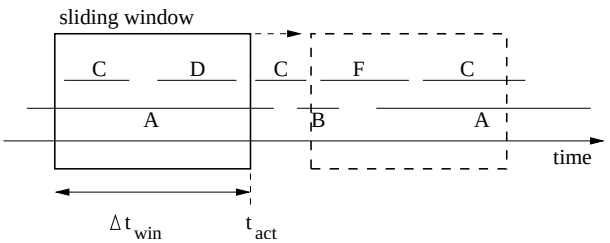


Figure 3.7: Sliding a window of width Δt_{win} along the sequence.

To be able to classify or distinguish a set of labeled intervals as a certain pattern, we have to compare the interval borders with each other. For instance, to distinguish "*A starts B*" from "*A overlaps B*" it is sufficient to observe A a little before B, but it does not play a role when A has started exactly. Therefore the exact length of the

very first and last state interval is not important for the classification of the whole set. In general, if we write down the initial and final points of all the participating state intervals in ascending order (cf. figure 3.8), then skip the very first (t_{min}) and last one (t_{max}) we obtain the length of the pattern by subtracting the latest ($t_{nearmax}$) from the earliest point shifted by the window width ($t_{nearmin} + \Delta t_{win}$). If some interval borders are identical (as in "A is-finished-by B") then we have to write them down multiple times. If this happens for the very first or last point in time, we remove only one occurrence (in "A is-finished-by B" we obtain the duration of B plus the window width as the pattern length).

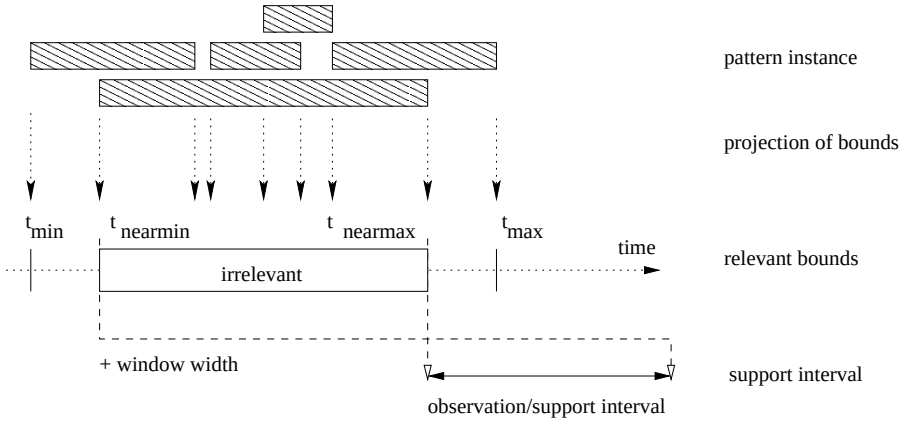


Figure 3.8: Observation interval.

Figure 3.9 shows the algorithm to calculate the observation interval of a pattern instance. If the sliding window reference point t_{act} lies within $[t_0, t_1]$, then the pattern can be observed within the window (of width Δt_{win}). Note that it may happen that the algorithm returns values $t_1 < t_0$ (then $[t_0, t_1] = \emptyset$), in this case the pattern cannot be observed within a window of width Δt_{win} (but would be observable in a window of larger width).

Theorem 2 *The time complexity of algorithm 3.9 is $O(d)$, where d is the size of the pattern instance.*

Proof of Theorem 2: Actually, we do not have to sort the interval bounds, since we are only interested in a fixed number of elements in the linear ordering, namely the two first and two last elements. The subroutine *determine_min* is $O(1)$ and is called $2d$ times in lines 4-7. Since the pattern has normalized form, the t_{min} and $t_{nearmin}$ values can be obtained even simpler: From (3.5) we know that these two values are among b_1, f_1, and b_2 (where $[b_1, f_1)$ and $[b_2, f_2)$ denote the first two intervals of the pattern). Thus, $t_{nearmin}$ is either f_1 or b_2. ■

Since any subpattern P of Q occurs within Q, the number of P-occurrences is greater or equal to the number of Q-occurrences. Obviously the same is true for the periods of observation of the respective patterns:

Theorem 3 *With an increasing pattern complexity the size of the observation interval cannot increase:*

$$\forall \textit{patterns } P, Q: \quad Q \sqsubseteq P \;\Rightarrow\; \text{supp}(Q) \geq \text{supp}(P) \tag{3.8}$$

```
1  proc observation_interval(→ d, → lower[], → upper[], → Δt_win, ← t_0, ← t_1)
2    if d = 1 then t_0 ↤ lower[1]; t_1 ↤ upper[1] + Δt_win; return fi
3    t_max ↤ −∞; t_nearmax ↤ −∞;
4    for i ↤ 1 to d do
5        determine_max(lower[i], t_max, t_nearmax);
6        determine_max(upper[i], t_max, t_nearmax);
7    od
8    t_nearmin ↤ min{upper[1], lower[2]}
9    t_0 ↤ t_nearmax; t_1 ↤ t_nearmin + Δt_win;
10 .
12 proc determine_min(→ t, ↔ t_min, ↔ t_nearmin)
13   if t < t_min then t_nearmin ↤ t_min; t_min ↤ t;
14   elsif t < t_nearmin then t_nearmin ↤ t; fi
15 .
17 proc determine_max(→ t, ↔ t_max, ↔ t_nearmax)
18   if t > t_max then t_nearmax ↤ t_max; t_max ↤ t;
19   elsif t > t_nearmaxt_nearmax ↤ t; fi
20 .
```

Figure 3.9: Algorithm returns observation interval $[t_0, t_1]$ of a pattern instance of dimension d in a window of width Δt_{win}, where the i^{th} interval is given by $[lower[i], upper[i]]$.

One disadvantage of this definition of pattern visibility may be that we cannot observe a pattern that consists out of two state intervals, each of both with a duration larger than the window width Δt_{win}. If such patterns turn out to be important we should reconsider the selection of Δt_{win} or we have to relax the observability conditions. For example we might alternatively require that all intervals must be observable – but not necessarily all bounds. In this case we would assume that the bounds are clear from the context or (as far as left bounds are concerned) will be *remembered* by the operator (in a monitoring and control application). However, we do not consider this modification here.

Later in section 5.2 the determination of the observation interval will be revisited.

3.2.2 Finding Temporal Patterns in Interval Sequences

When extracting labeled intervals from time series, the sequence of intervals will be ordered since the time series is usually ordered. Therefore we assume that the sequence of labeled intervals is sorted lexicographically by initial time, final time, and label. When building temporal patterns (s, R) from labeled intervals (b_k, f_k, s_k) in the sequence, the pattern (s, R) is automatically normalized if we preserve the order of the sequence (Theorem 1), that is: for any sorted list of indices $k_1 < k_2 < ... < k_n$ the temporal pattern defined by

$$s(i) = s_{k_i}, \qquad R[i,j] = \mathrm{ir}([b_{k_i}, f_{k_i}], [b_{k_j}, f_{k_j}])$$

is already normalized.

Using normalized temporal patterns also simplifies testing for the subpattern relation. To decide whether $P \sqsubseteq Q$ holds (where Q will be the content of the sliding window) we have to check if there is a state embedding π satisfying (3.2): For each state $s_P(i)$ of P we have to check whether there is an index j such that

$s_Q(j) = s_P(i)$. With non-normalized patterns, for each i we would have to try every possible value for j, that is $1 \leq j \leq \dim(Q)$. But in normalized form, the embedding π is always strictly increasing to preserve the temporal ordering of the state intervals. This fact simplifies the naive algorithm in figure 3.10, but nevertheless the subrelation check has to implement a backtracking mechanism, since a state mismatch in higher indices may only be resolved by a different assignment at lower indices.

```
 1 funct subrelation_check(→ P, → Q, ← π)                    P and Q normalized
 2   if dim(P) > dim(Q) then return false fi;
 3   π(·) ↤ 0;
 4   return perm(1, 1, P, Q, π)
 5 .
 7 funct perm(→ i, → j, → P, → Q, ↔ π)                       i/j current index in P/Q
 8   found ↤ false
 9   repeat
10     if s_P(i) = s_Q(j)
11       then π(i) = j; ok ↤ true;
12            for k = 1..i do ok ↤ ok ∧ R_P[i, k] = R_Q[j, π[k]] od
13            if ok then
14                 if i < dim(P)
15                   then found ↤ perm(i + 1, j + 1, P, Q, π)
16                    else found ↤ true, π is a solution
17                 fi
18            fi
19     fi
20     j ↤ j + 1;
21   until (j > dim(Q)) ∨ (found)
22   return found;
23 .
```

Figure 3.10: The subrelation check. The function *subrelation_check* takes two patterns P and Q and returns **true** iff $P \sqsubseteq Q$. It also yields the state mapping π (cf. definition 3). To find all possible instances of the pattern, remove "$\vee(found)$" from line 21.

Backtracking can be computationally very expensive if the number of intervals increases. While the theoretical number of possible state assignments is $\binom{\dim(Q)}{\dim(P)}$, most of the embeddings are not valid since a state $s_P(i)$ has to be mapped to an identical state $s_Q(\pi(i))$ etc. With respect to correct state mapping the worst case occurs when all states $s_P(i)$ and $s_Q(j)$ are identical – then every embedding is hypothetically applicable. But in this case, due to the maximality assumption, only the temporal relationship *before* remains valid such that no backtracking is necessary, any $\dim(P)$ states of Q will match pattern P. The time complexity to find a match is in this special case $O(\dim(P) + \frac{1}{2}\dim(P)^2) = O(\dim(P)^2)$ – loop over the first $\dim(P)$ states in Q and compare $\frac{1}{2}\dim(P)^2$ interval relationships.

However, in the general case one can easily construct pathological situations where the algorithm exploits its exponential worst case costs. Consider the pattern "*A before B, B meets C*" in the context of figure 3.11(a). The backtracking algorithm will try every occurrence of A and for each assignment every occurrence of B, just to realize then that there is no C in a *meets* relationship. Of course, there is a number of very simple tests that immediately come up to our mind to avoid the backtracking in situations like this. For instance, one test is to check if at least

all states and interval relationships are present in the sliding window.

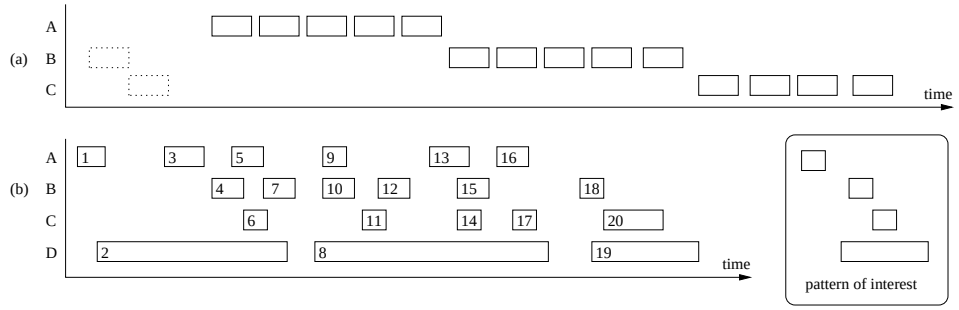


Figure 3.11: Finding subpatterns.

Although these simple tests can be very effective, one can easily find examples where they do not help: If we additionally consider the dotted intervals in figure 3.11(a) the test for an occurrence of an B *meets* C relationship is positive – it does not prevent us from performing intensive backtracking. Therefore we propose a more general approach to efficient subpattern testing. The idea is to concentrate the efforts on those relationships that seem to be critical in the sense that there is only a small number of occurrences. For instance, if we have a large number of "A *before* B" relationships but only a very small number of "B *meets* C" relationships, we try to fix the (B, C) pair first in order to reduce the potential cost of future backtracking. We still may have to explore many possible combinations, but rather than selecting the states in the order given by the normalized form, we spend some effort on finding the state with the smallest number of possible combinations. Then, on the average, proving that no instance is present is computationally less expensive, since the average backtracking depth is reduced.

Since we will match many different patterns P against the content of the same pattern Q (the sliding window), we employ a different representation for Q: Instead of storing the n^2 ($n = \dim(Q)$) interval relationships in a matrix, we use a more sophisticated data structure: Any pair of state intervals is specified by a point in $\mathbb{N}^2$, where the numbers denote the indices in the lexicographically sorted sequence. We define a map $\omega : \mathcal{S} \times \mathcal{I} \times \mathcal{S} \to \mathcal{P}(\mathbb{N}^2)$ which yields for given labels A, B and an interval relationship r a set of instances such that each element $(i_1, i_2) \in \omega(A, r, B)$ satisfies: $s_{i_1} = A$, $s_{i_2} = B$ and $R[i_1, i_2] = r$. This representation requires more organizational overhead, however, we still have $\left|\bigcup_{A\in\mathcal{S}} \bigcup_{r\in\mathcal{I}} \bigcup_{B\in\mathcal{S}} \omega(A, r, B)\right| = n^2$, that is, the space complexity is not increased. While scanning through the sequence, the representation can be updated incrementally.

Let us illustrate the algorithm by an example. Consider the problem in figure 3.11(b). When checking for $P \sqsubseteq Q$ we construct a matrix I of lists from the i-function: $I[i, j] = \omega(s_P(i), R_P[i, j], s_P(j))$. Since R_P is pseudo-symmetric, the same is true for I. In case of figure 3.11(b) we have

$$I[1,2] = \omega(A, \mathit{before}, B) = \{(1,4),(1,7),(1,10),..,(3,4),(4,7),..\} \quad (3.9)$$
$$I[1,3] = \omega(B, \mathit{meets}, C) = \{(4,6),(18,20)\} \quad (3.10)$$
$$I[1,4] = \omega(A, \mathit{before}, D) = \{(1,8),(1,19),(3,8),(3,19),(5,8),..\} \quad (3.11)$$
$$I[2,3] = \omega(B, \mathit{during}, D) = \{(4,21),(10,8),(12,8),(15,8)\} \quad (3.12)$$

The cardinality of the lists in the matrix is

$\lvert I\rvert$	A	B	C	D
A		21	21	9
B			2	4
C				5
D				

Here, the relationship B *meets* C is most restrictive, since there are only 2 state pairs that match this relationship. Therefore, we decide to start the search with fixing the (B, C) pair and have therefore a maximal number of 2 backtracks in this level. We start with the first possibility $(4, 6)$ (cf. (3.10)), which means that we define π partially by assigning interval #4 to the B state and interval #6 to the C state: $\pi(2) = 4$, $\pi(3) = 6$. Before we enter the next recursion to further fix π, we update the content of the matrix I according to our choices: we remove all entries

$$\begin{array}{l}(i', j') \in I[k, \cdot] \text{ if } i' \neq \pi(k) \quad \text{and} \\ (i', j') \in I[\cdot, k] \text{ if } j' \neq \pi(k)\end{array} \tag{3.13}$$

for any state k that has been fixed to $\pi(k)$. The pairs in $I[1, 2]$ contain possible assignments for the A *before* B relationship, however, since we have fixed B by $\pi(2) = 4$, we can remove all pairs $(i_1, i_2) \in I[1, 2]$ with $i_2 \neq 4$. The same is true for pairs in $\omega(A, \textit{before}, C)$, $\omega(B, \textit{during}, D)$, and $\omega(C, \textit{during}, D)$. This leads us to the following updated cardinalities:

$\lvert I'\rvert$	A	B	C	D
A		2	2	9
B			$\cdot$	1
C				1
D				

Since we have fixed B and C, that is $\pi(2)$ and $\pi(3)$, the pairs in $\omega(A, \textit{before}, D)$ are not affected, since they specify $(\pi(1), \pi(4))$. Nevertheless, for any fixed state k and pair $(i', j') \in I[i, j]$ (this includes those in $I[1, 4] = \omega(A, \textit{before}, D)$) we must have

$$\begin{array}{l}R_P[\min(k, i), \max(k, i)] = R_Q[\min(\pi(k), i'), \max(\pi(k), i')] \quad \text{and} \\ R_P[\min(k, j), \max(k, j)] = R_Q[\min(\pi(k), j'), \max(\pi(k), j')]\end{array} \tag{3.14}$$

to satisfy the temporal constraints of the pattern. In our particular example for the fixed states $k \in \{2, 3\}$ we have for $(i_1, i_2) \in I[1, 4]$:

$$\begin{aligned} R_P[1, 2] &= R_Q[\pi(1), \pi(2)] = R_Q[i_1, 4] \\ R_P[1, 3] &= R_Q[\pi(1), \pi(3)] = R_Q[i_1, 6] \\ R_P[2, 4] &= R_Q[\pi(2), \pi(4)] = R_Q[4, i_2] \\ R_P[3, 4] &= R_Q[\pi(3), \pi(4)] = R_Q[6, i_2] \end{aligned}$$

This simply means that any potential A state has to satisfy the *before* relationship to B and C, and any potential D state has to satisfy the *during* relationship to B and C. If we update the lists once more we obtain

$\lvert I''\rvert$	A	B	C	D
A		2	2	0
B			$\cdot$	1
C				1
D				

From the 0 entry we can see that there is no valid (A, D) pair that matches our choice of (B, D), therefore we withdraw our assumption and now test for $(18, 20)$. Immediately, we obtain $|I(B, \mathit{during}, D)| = 0$ and conclude that $P \not\sqsubseteq Q$ since there are no further possibilities for (B, C).

In comparison to algorithms 3.10, after every state assignment we have to check all list entries in the I matrix (maximal number of $n \cdot (n-1)/2$ entries), which is $O(n^2)$ in contrast to the row/column check in algorithm 3.10, which is $O(n)$. On the other hand, we actively influence the maximum backtracking depth.

3.3 Enumerating Frequent Temporal Patterns

Now that we are able to construct patterns and estimate their frequency, we are interested in enumerating all *frequent* patterns. Since we will induce rules from the patterns we have found and thus generalize from examples, we are not interested in patterns that occur only a few times (because or belief in such a rule would be very low). Therefore we enumerate only those rules $P \rightarrow R$ whose support $\mathrm{supp}(R)$ is greater or equal to an arbitrarily chosen lower bound supp_{min}.

In a naive implementation we could enumerates all possible patterns and check each individually for being frequent. The feasibility of such an approach depends on the size of the space of temporal patterns. Suppose we have n frequent patterns of size $k = 1$, how many possibilities of 2-candidates are there? In contrast to itemsets in traditional association rule mining, where the same item can occur only once in a set, a state can occur multiple times in a pattern like "A before A". But due to the maximality assumption (3.1) patterns like "A overlaps A" cannot occur, thus we have only n patterns of the kind "A before A" with identical states. For normalized patterns we have 7 temporal relationships, in 6 out of 7 the order of the labels is important, but not with *equals* ("A equals B" is the same as "B equals A"). Thus, for *equals*-patterns we have $\binom{n}{2}$ combinations and for the remaining 6 relationships we have $\frac{n!}{(n-2)!}$ combinations. This gives us a total number of $n + \binom{n}{2} + 6\frac{n!}{(n-2)!}$ 2-candidates.

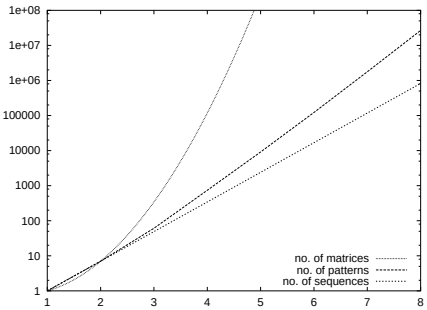


Figure 3.12: Number of distinct normalized temporal patterns (log scale) versus the size k of the temporal patterns (assuming k distinct labels).

As k increases the number of possible temporal patterns grows exponentially, as shown in figure 3.12. (The other curves show the number of matrices (arbitrary selection of upper triangular matrix, regardless whether it is a valid temporal pattern or not) and the number of "sequential patterns" where only a single relationship to the left neighbour of each interval is maintained.) For $k = 7$ we have already more

than 1.75 million different interval relationship matrices. For the figure we have just counted the number of different relationship matrices of size $k \in \{1, .., 8\}$, but we do not have considered label selection and/or permutation yet. If there are n different labels ($n > k$) to select from, the number of possible selections $n!/(n-k)!$ (ordering is relevant) has to be multiplied with the number shown in figure 3.12. For a subset of the patterns, we could also select partially identical labels (as in "*A before A*"). This gives us an impression of the tremendous number of possible temporal patterns.

In association rule mining [Agrawal et al., 1996], which became popular in market basket analysis, rules between sets of (purchased) items are searched, e.g. "if the customer buys milk and bread, then in 80% she or he will also buy butter". While the number of possible patterns is much smaller when dealing with itemsets only, the approach to pattern enumeration has proven to be efficient and will be adopted for our purposes in the following sections. An algorithm is given in figure 3.13; at this level of detail the algorithm holds for itemsets as well as temporal patterns. To find all frequent patterns we start in a first database pass with the estimation of the support of every single state (also called candidate 1-patterns). After the kth run, we remove all candidates C_k that have missed the minimum support and create out of the remaining frequent k-patterns F_k a set of candidate $(k+1)$-patterns whose support will be estimated in the next pass. This procedure is repeated until no more frequent patterns can be found. The fact that the support of a pattern is always greater or equal to the support of any of its subpatterns (Theorem 3) guarantees that we do not miss any frequent patterns.

```
1  proc find_freq_patterns(→ S, → (I_i)_i, ← (F_i)_i)
2    k ↤ 1
3    C_k ↤ {s ∈ S}                                   1-candidates
4    repeat
5      support_estimation(C_k, (I_i)_i)              pass over sequence (I_i)_i
6      F_k ↤ {P ∈ C_k | supp(P) > supp_min}          freq. k-patterns
7      C_{k+1} ↤ candidate_generation(F_k)           build (k+1)-candidates
8      k ↤ k + 1
9    until C_k = ∅                                   until no new candidates
10 .
```

Figure 3.13: High level description of frequent pattern enumeration algorithm. The input parameters are the set of symbols $\mathcal{S}$ and the labeled interval sequence $(I_i)_i$, the output is the sequence of frequent pattern sets.

Any frequent k-pattern must have k frequent $(k-1)$-subpatterns, each of these $(k-1)$-subpatterns must have $k-1$ frequent $(k-2)$-subpatterns and so forth. If there is only a single frequent k-pattern, we will have to discover $k!$ frequent patterns of size $\leq k$. (For more than one frequent k-pattern several subpatterns will probably be shared.) Since we do not want to restrict the maximal size k of the patterns, the number of frequent pattern may explode quickly. But since the number of symbols is finite and the interval lengths are not arbitrary small, there is only a finite number of frequent pattern we will observe within a window of fixed width. The number of frequent patterns is greatly influenced by the minimum support threshold $\min_{supp}$, which can be increase to keep the number of frequent patterns manageable.

There are two major subtasks in algorithm 3.13, (a) the generation of new $(k+1)$-candidates from the set of frequent k-patterns and (b) the support estimation of candidate patterns to determine whether candidates are frequent or not. These two

steps will be discussed in the following sections.

3.3.1 Candidate Generation

The number of patterns grows exponentially with their size k, but we can expect most of them to be infrequent. However, efficient pruning techniques are necessary to consider as few candidate patterns as possible during support estimation. We use a cascade of three different pruning techniques to achieve this.

Probably the most important pruning technique is the one that is also used for the discovery of association rules [Agrawal et al., 1996]. It is particularly important because it can be used to effectively construct the set of candidates. Due to (3.8), every k-subpattern of a $(k+1)$-candidate must be frequent, otherwise the candidate itself cannot be frequent. Thus, to be a $(k+1)$-candidate, all k-subpatterns (that are obtained by removing one interval) must be contained in the set of frequent k-patterns. Any of these k-subpatterns specifies the (unknown) $(k+1)$-candidate up to a label and one row and column of the relationship matrix. The missing information can be taken from another frequent k-subpattern of the candidate. We will use the two k-subpatterns P and Q where one of the last two intervals of the candidate is missing, thus P and Q share a common $(k-1)$-pattern as a prefix. Joining P and Q yields the candidate itself (as will be discussed below). Thus, to enumerate as few non-candidate $(k+1)$-patterns as possible, it is sufficient to join any two frequent k-patterns. We will not miss any frequent candidates, because due to (3.8) the two subpatterns P and Q must be in the set of frequent k-patterns, otherwise the candidate itself cannot be frequent.

Let us denote the remaining states in P and Q besides those in the prefix as p and q respectively. We denote the interval relationship between p and q in the candidate pattern $X = (s_X, R_X)$ as $R_X[k, k+1] = r$. Figure 3.14 illustrates how to build the $(k+1)$-pattern matrix R_X out of R_P and R_Q. Since R_P and R_Q are identical with respect to the first $k-1$ states in normalized form, the same is true for the new pattern X (indicated by the same submatrix A). The relationship between p and q and the first $k-1$ states can also be taken from R_P and R_Q. Thus, as we can see in figure 3.14(c), the only degree of freedom is r. From the $(k-1)$-pattern prefix and the two states p and q we thus can build up a $(k+1)$-pattern which is completely specified up to the relation between p and q.

The freedom in choosing r yields 13 different patterns that might become candidate $(k+1)$-patterns, because there are 13 possible interval relationships. Since we can restrict ourselves without loss of generality to normalized patterns, the number of possible values for r reduces to a maximal number of 7. Once we have fixed r the construction of X is finished and we can check whether all remaining k-subpattern of X (those $k-1$ patterns where one of the first $k-1$ intervals is removed) are also frequent. If one is not, X cannot be a frequent $(k+1)$-pattern and X does not become a candidate.

Before we apply this pruning step to each of the seven $(k+1)$-patterns, we apply another pruning technique based on the law of transitivity. For example, the two 2-patterns "A meets B" and "A meets C" share the primitive 1-pattern "A" as a common prefix. We have to fix the missing relationship between B and C to obtain a 3-candidate. The law of transitivity for interval relations [Allen, 1983] tells us that the possible set of interval relations is $\{$*is-started-by*, *equals*, *starts*$\}$. In normalized form, only 2 out of 7 possible relationships remain. In general, for each state $s(i)$ of the first $k-1$ states we apply Allen's transitivity table to the relationship between p and $s(i)$ ($R_P[k, i]$) and $s(i)$ and q ($R_Q[i, k]$). Only those values for r that do

	R_P	s_P $0 \ldots k-1$	p
	0		
s_P	$\vdots$	A	B
	$k-1$		
	p	C	$=$

(a) Pattern P

	R_Q	s_Q $0 \ldots k-1$	q
	0		
s_Q	$\vdots$	A	D
	$k-1$		
	q	E	$=$

(b) Pattern Q

	R_X	s_X $0 \ldots k-1$	p	q
	0			
s_X	$\vdots$	A	B	D
	$k-1$			
	p	C	$=$	r
	q	E	$\bar{r}$	$=$

(c) New pattern X

Figure 3.14: Generating a candidate $(k+1)$-pattern X out of two k-patterns P and Q that are identical when restricted to the first $k-1$ states. The inverse interval relationship of r is denoted by $\bar{r}$.

not contradict the results of the $k-1$ applications of the transitivity table yield a candidate pattern.

The step from $k = 1$ to $k = 2$ is in some sense a "worst case" because in case of 1-patterns we cannot conclude anything about the interval relationship r via the law of transitivity. However, as k becomes larger, the possible values of r can be reduced significantly.

Finally, for every temporal pattern Q we maintain an *observed* and *expected support set* O_Q and E_Q, resp. The set O_Q contains all points in time that contribute to the support of the pattern Q, that is, all window positions t_{act} in which the pattern can be observed in the sliding window (cf. definition 5). Before we consider a $(k+1)$-pattern P as a candidate pattern, we intersect[2] all sets O_Q of all k-subpatterns Q of P. The result gives us the potential support of P in E_P. The accumulated length of E_P serves as a tighter upper bound of the support of P than $\min\{E_Q \,|\, Q \sqsubseteq P, \dim(Q) = k\}$ does. If it stays below supp_{min} the pattern cannot become a frequent pattern, therefore we do not consider it as a candidate.

Candidate Generation Algorithm. To perform pruning we have to search the k-subpatterns of a $(k+1)$-candidate in the set of frequent k-patterns. Thus we need a sequential order on temporal patterns. Transforming a pattern $P = (s, R)$ into a

[2]The sets O_Q and E_Q can be organized as lists of intervals. The intersection is also a list of intervals. We only have to add up the interval lengths to obtain the accumulated length.

vector

$$\Big(\underbrace{\underbrace{\underbrace{s(1)}_{1\text{-pattern}}}_{2\text{-pattern}}}_{3\text{-pattern}}, s(2), R[2,1], s(3), R[3,1], R[3,2], ...\Big)$$

and using lexicographical comparison on the tuples employs such a total order. From the tuples the original patterns P can be reconstructed. During candidate generation, however, we only compare patterns of identical sizes, leading to tuples of identical dimension. Within a sorted list blocks of frequent k-pattern with identical $(k-1)$-prefix can be identified easily (cf. [Mannila et al., 1997]). Two patterns belong to the same block if they differ only in the last state (label and interval relationships), which corresponds to a difference in the last k dimensions of the tuple. By B_P we denote the index of the first pattern in the list that belongs to the same block as P does. If $B_P = B_Q$ both pattern P and Q belong to the same block, we enter a new block if $B_P \neq B_Q$ for consecutive patterns P and Q in the list. For the first pruning technique we have to search for all $(k-1)$-subpatterns of the newly generated $(k+1)$-pattern. Only $k-1$ out of $k+1$ patterns have to be searched, 2 were already used to build the new candidate. The search for the remaining subpatterns can be done efficiently using binary search in the sorted field of frequent patterns.

Figure 3.15 shows the algorithm for candidate generation. Making use of the block start B_P, any two patterns with identical $(k-1)$-prefix can be identified easily (loops in lines 3 and 5). For primitive 1-patterns P we have $B_P \equiv 1$. New candidate patterns C will be generated in accordance with the pattern order (such that no explicit sorting becomes necessary) and the values B_C are also set during candidate generation (lines 2, 4 and 14). The application of the law of transitivity for the second pruning technique (line 7) is very cheap in terms of complexity ($k-1$ table look-ups). With the third pruning technique, the intersection of sets of intervals (support sets) is linear in the number of intervals, however, the actual number of intervals is difficult to estimate and is significantly influenced by the overall number L of state intervals in the sequence and the window width Δt_{win}.

Theorem 4 *The time complexity of algorithm 3.15 is*

$$O(k \cdot |F_k| \cdot |\mathcal{S}| \cdot (k^2 + \log|F_k| + k \cdot L)).$$

Proof of Theorem 4: The iteration over all candidate pairs with identical $(k-1)$-prefixes is $O(|F_k|M)$ where M is the maximum block size, which is bounded by $7|\mathcal{S}|$ (the outer loop in line 3 is iterated $|F_k|$ times and the inner loop in line 5 is iterated $M \leq 7|\mathcal{S}|$ times. The candidate pattern X is build in $O(k+k^2)$ and the $k-1$ remaining k-subpatterns of X are build in $O((k-1)(k+k^2)) = O(k^3)$. The $k-1$ subpatterns S_l have to be searched in F_k, which can be done in $O((k-1)\log|F_k|)$ using binary search. Next, for each of the possible 7 interval relationships in a normalized pattern (line 8) the law of transitivity is applied $k-1$ times (line 7) in $O(7(k-1))$. The length of any O_P is bounded by the number L of intervals in the sequence (more precisely: by the maximum number of intervals in the sequence with the same label). The intersection of two interval sequences of length L is done in $O(2L)$, therefore line 11 is done in $O((k+2)L)$. The sum of interval lengths of E_X is also obtained in $O(L)$. This gives us the overall complexity

$$O(\underbrace{|F_k|}_{\text{loop l. 3}} \cdot \underbrace{|\mathcal{S}|}_{\text{loop l. 5}} \cdot(\underbrace{k^3}_{\text{cand. gen.}} + \underbrace{k\log|F_k|}_{\text{subp. search}} + \underbrace{k^2 L}_{\text{intersect}}))$$

```
1  proc candidate_generation(→ F_k, → (O_P)_{P∈F_k}, ← C_{k+1}, ← (E_P)_{P∈C_{k+1}})
2    C_{k+1} ↤ ∅; n ↤ 0;                                   n is actual value of |C_{k+1}|
3    for i ↤ 1 to |F_k| do
4        thisblock ↤ n + 1; j ↤ B_{F_k[i]};
5        while B_{F_k[i]} = B_{F_k[j]} do                  F_k[i] and F_k[j] in same block
6              construct X from F_k[i] and F_k[j] acc. to fig. 3.14 (up to r);
7              R ↤ {r ∈ I | X satisfies law of transitivity with R_X[k, k − 1]};
8              for r ∈ R do
9                  build k-subpatterns S_l of X (besides F_k[i] and F_k[j]);
10                 if (∀1 ≤ l ≤ k − 1 : S_l ∈ F_k)
11                   then E_X ↤ O_{F_k[i]} ∩ O_{F_k[j]} ∩ O_{S_1} ∩ ... ∩ O_{S_{k−1}};
12                        if (card(E_X) > supp_min)
13                          then n ↤ n + 1; C_{k+1}[n] ↤ X;
14                               B_{C_{k+1}[n]} ↤ thisblock;
15                        fi
16                 fi
17             od
18             j ↤ j + 1;                                  next pattern in block
19       od
20   od
21 .
```

Figure 3.15: High-level description of candidate generation. The procedure takes the set of frequent k-patterns F_k together with the observed support sets O_P and yields a set of candidate $(k+1)$-patterns C_{k+1} together with the potential support sets E_P.

■

The complexity of algorithm 3.15 is dominated by $|F_k|$ and L. In practice, while the sequence length L may become very large (size of the database), the actual complexity of the support interval intersection does not really depend on L but the fragmentation of the support (number of support intervals in O_P for a pattern P). When starting with primitive 1-patterns, support intervals get merged if their (Hausdorff) distance is below the window width (compare with figure 3.6(d)). With frequent patterns, O_P consists quite often of a very small number of large support intervals. When the patterns get more complex, the total support decreases but the fragmentation of the set O_P may increase. For higher values of k the complex patterns occur rarely and the number of support intervals becomes identical to the number of pattern instances. The number of pattern instances finally approaches zero when the minimum support is no longer reached by the pattern. Therefore, the average complexity varies depending on k, having local minima at $k = 0$ and $k = k_{max}$, reaching a local maximum somewhere in between (by k_{max} we denote the maximum size k of a frequent pattern found by algorithm 3.13).

3.3.2 Support Estimation

In order to estimate the support of the candidate patterns, we scan through the sequence and keep track of all intervals (and their relationships) that are currently visible in the sliding window. In the following, we denote the content of the sliding window as a normalized pattern W.

In the first database pass, we build up the set of states $\mathcal{S}$ and simultaenously the sets of observed support $O_{\{s\}}$, $s \in \mathcal{S}$. This is done in $O(L)$, where L denotes the number of labeled intervals in the analyzed sequence. While deciding whether a 1-pattern occurs in the sliding window or not is trivial[3], checking k-candidate patterns with $k > 1$ becomes computationally more expensive (as we have seen in section 3.2.2). Whenever the right or left bound of the sliding window meets an interval bound, the changing content of the sliding window may require some action since temporal relationships get resolved or lost, or new intervals are introduced in the window. In principle, whenever the content of the window changes we have to check every candidate whether it is a subpattern of the sliding window or not. Since the size of C_k may become very large, we are again in need of pruning mechanisms that prevent us from unnecessarily calling the subpattern checking routine.

Therefore, the set of candidate patterns is partitioned into three subsets, which we call the set of passive, active, and potential candidates. Recall that by t_{act} we denote the right bound of the sliding window. The set of passive candidates contains those candidates P that we do not expect in the current sliding window because the potential support set E_P does not contain the time of the current window position, that is, $t_{act} \notin E_P$. The set of potential candidates contains those candidates for which we have $t_{act} \in E_P$, that is, there is a chance of observing P in the window and it is worth checking P against the sliding window. Finally, the set of active patterns contains those patterns that are currently observable in the sliding window, that is $t_{act} \in O_P$.

The rationale behind this subdivision is the following:

Passive patterns P do not have to be checked against $P \sqsubseteq W$. For any passive pattern we know that one of its subpatterns is currently not observable, therefore the pattern itself cannot be visible.

Active patterns have been observed in the (recent) past, so it is not necessary to check them again as long as the instance found before is still visible because our notion of support is independent of the number of instances simultaneously visible. Since we can compute in advance how long a given pattern instance will be visible, it is possible to postpone the next call of the pattern matching subroutine for P unless the instance disappears.

Potential patterns are the only patterns for which we have to check continuously whether a new instance can be found in the window.

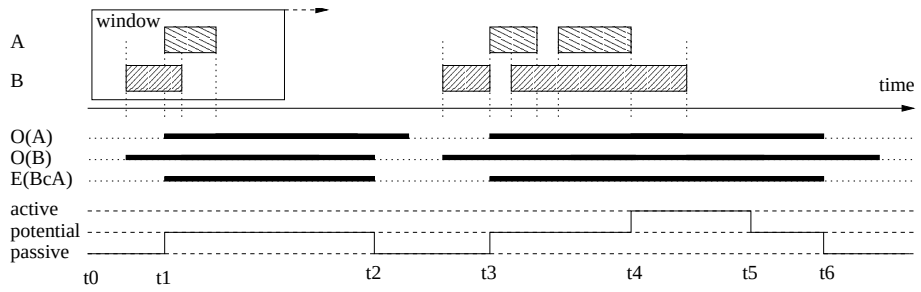


Figure 3.16: History of belongingness to the sets of passive, potential, and active patterns of an example pattern "*B contains A*".

[3]In fact, for a single state s we do not even have to build up the sliding window, it is sufficient to unify all intervals $[b_i, f_i + \Delta t_{win}]$ for every (b_i, f_i, s) in the sequence.

Figure 3.16 traces an example pattern $P =$"*B contains A*" through the support estimation. In the figure, by $O(A)$ and $O(B)$ the set of observed support O_A and O_B are depicted. During candidate generation the set of potential support E_P (shown as E(BcA) in the figure) has been generated. The solid line in the lower three rows indicate to which class P belongs. At the beginning (t_0), the pattern is passive. From the observation of A and B we know that P might be observable at t_1, since both subpatterns are visible from then on. Therefore, P becomes a potential pattern at t_1. Again from the intersection of O_A and O_B we know that at time t_2 we cannot observe A and B any longer, therefore at t_2 the pattern P is removed from the set of potential patterns and falls back into the set of passive patterns. We schedule the reconsideration of P at time t_3, where the next interval in E_P starts. This time we observe an instance of P at t_4. We calculate the observation duration and let P become an active pattern. From the observation duration of P we know that this instance will vanish at t_5, any further checking of P against the sliding window is postponed until t_5. At t_6 the potential support interval in E_P terminates, which causes P to fall back to the set of passive patterns. The intervals for which P became an active pattern then become the set of observed support O_P, which will be used by the candidate generation in the next loop.

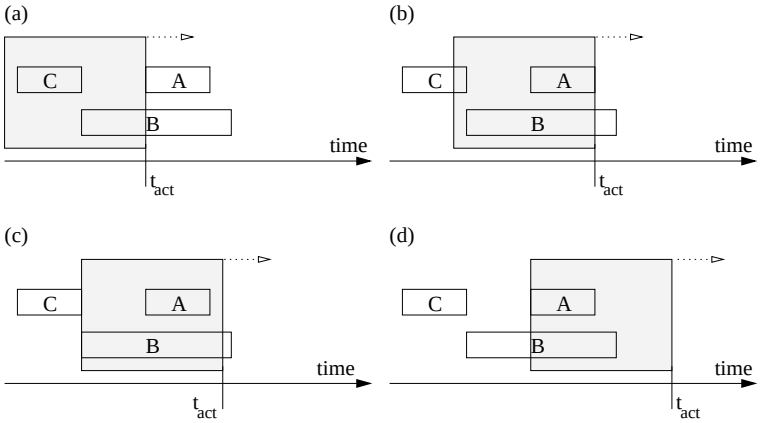


Figure 3.17: Important events when sliding the window along the sequence.

Let us discuss in greater detail in which situations a pattern moves from one set of the partition into another. We speak of a Leave-Passive event if a pattern is removed from the set of passive patterns to the set of active or potential patterns. In the same way we define Leave-Active and Leave-Potential events. As already discussed, if interval and window bounds coincidence this will cause such events. There are four possible situations, as depicted in figure 3.17:

Case (a) *right bound of sliding window meets left bound of interval*: In this situation a new interval enters the sliding window, we speak of a New-Interval event. One or more Leave-Potential events may be caused by the New-Interval event since potential patterns may become active. In the depicted example the pattern "*C before A*" becomes active.

Case (b) *right bound of sliding window meets right bound of interval*: An interval becomes completely visible, which is called a Full-Interval event. Again, one or more Leave-Potential events may be caused by the Full-Interval event, since the temporal relationship to other intervals is resolved. In the depicted example the pattern "*B contains A*" becomes active; before this window position we were not sure about the exact relationship, a "*B overlaps A*"

pattern was also possible.)

Case (c) *left bound of sliding window meets right bound of interval*: One or more LEAVE-ACTIVE events may be caused. In the example, the pattern "*C before A*" disappears from the sliding window.

Case (d) *left bound of sliding window meets left bound of interval*: Similar to case (c), one or more LEAVE-ACTIVE events may be caused in this situation. In the example, the pattern "*B contains A*" disappears from the sliding window.

Cases (a) and (b) deal with the appearance and (c) and (d) with the disappearance of pattern instances, since in (a) and (b) new interval bounds *enter* the window but no interval bounds *vanish* (and with (c) and (d) we have the reverse). Once a new interval has encountered the sliding window (NEW-INTERVAL event), its length implicitly defines the associated FULL-INTERVAL event. Cases (c) and (d) are not necessarily relevant with every interval, but only if the interval is currently used by a detected pattern instance. Once we have detected an instance, from its observation interval we also know in advance when we will have to reschedule the next check for this pattern – these points in time correspond to cases (c) and (d). A found instance therefore implicitly defines a LEAVE-ACTIVE event.

Figure 3.18 illustrates the pattern state transitions. Initially, all patterns are passive. The intervals in E_P are organized as a stack, the actual top element is denoted by $[a_P, d_P]$, where a_P is the activation time of pattern P and d_P is the deactivation time. When the activation time a_P is reached, depending on whether an instance can be observed in the window or not, P becomes a potential or even active pattern. A potential pattern or active pattern falls back to the set of passive patterns if the deactivation time d_P is reached. These four transitions always occur at times a_P or d_P. A potential pattern becomes active as soon as a pattern instance is observed, which can only happen after a NEW-INTERVAL or FULL-INTERVAL event. From the observation interval of the instance we know when the instance vanishes. If we cannot find another instance at that point in time the pattern falls back into the set of potential patterns.

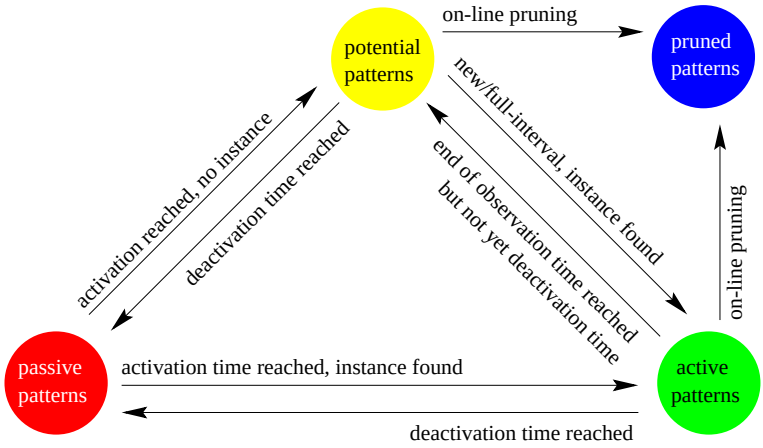


Figure 3.18: Pattern transition diagram.

Whenever a pattern instance has been found, the support of the pattern is updated incrementally, that is, we insert the period of pattern observation (the support) into O_P and remove the potential support interval $[a_P, d_P]$ from E_P. Since the sum of interval lengths in E_P is an upper bound of the remaining support, we can perform

a fourth online pruning test. If the support achieved so far ($\text{len}(O_P)$) plus the maximally remaining support ($\text{len}(E_P)$) drops below supp_{min} we do not consider the pattern any longer, it becomes a *pruned pattern*. At the end of each database pass, the set E_P is empty and O_P contains the support of P. All non-frequent patterns will be pruned.

```
 1 proc support_estimation(→ C, → (E_P)_{P∈C}, ← (O_P)_{P∈C})
 2   ACTIVE ↤ ∅; POTENTIAL ↤ ∅; PASSIVE ↤ C; t_act ↤ −∞;
 3   for all labeled intervals (b, s, f) in the sequence do        in lexicog. order
 4       schedule a NEW-INTERVAL event at time b;
 5       schedule a FULL-INTERVAL event at time f;
 6       while there are unprocessed events scheduled before b do
 7             e ↤ earliest event; t_act ↤ time of earliest event;
 8             e.P ↤ pattern associated with e; e.s ↤ state associated with e;
 9             if e = FULL-INTERVAL                          see also sec. 3.3.3
10               then process_all_potential(e.s);
11             elsif e = LEAVE-POTENTIAL
12                   then process_potential(e.P);
13             elsif e = LEAVE-PASSIVE
14                   then process_passive(e.P);
15             elsif e = LEAVE-ACTIVE
16                   then process_active(e.P);
17             fi
18             remove e from schedule;
19             shrink_window();
20       od
21       add_interval(b, s, f);                          process NEW-INTERVAL
22       process_all_potential(e.s);
23   od
24 .
```

Figure 3.19: The main loop of the support estimation algorithm. The procedure takes the set of candidate patterns C together with the potential support sets E_P and yields the observed support sets O_P.

Figure 3.19 depicts a high-level description of the support estimation algorithm. The algorithm iterates over the labeled intervals of the sequence (line 3) and schedules a NEW-INTERVAL and FULL-INTERVAL event. All events will be processed in the order of their temporal occurrence (lines 6-20). When it is time to process a

$$\left\{\begin{array}{r} \text{NEW-INTERVAL} \\ \text{FULL-INTERVAL} \\ \text{LEAVE-PASSIVE} \\ \text{LEAVE-POTENTIAL} \\ \text{LEAVE-ACTIVE} \end{array}\right\} \text{event, the affected} \left\{\begin{array}{l} \text{potential} \\ \text{potential} \\ \text{passive} \\ \text{potential} \\ \text{active} \end{array}\right\} \text{patterns}$$

have to be processed in order to keep the partition up to date. After each increment of t_{act} the content of the sliding window has to be updated. Since the sequence is assumed to be lexicographically ordered, the sliding window can be updated incrementally (adding intervals in line 21, removing intervals in line 19).

Theorem 5 *The support estimation algorithm in figure 3.19 can be implemented with time-complexity*

$$O(|C_k| \log |C_k| + (n+m) \cdot (T + \log |C_k|)), \tag{3.15}$$

where $n = \sum_{P \in C_k} \text{card}(E_P)$, $m = \sum_{i=1}^{L} |\textsc{Potential}_i|$, card *is the number of intervals in a set of intervals and* T *is the time complexity of the subpattern test for* k*-patterns.*

Proof of Theorem 5: To support efficient operations, the sets E_P and O_P are organized as sorted lists of intervals, the minimum is simply the left bound of the first interval in the list (and can be obtained in $O(1)$). We keep the set of passive patterns sorted by their activation time a_P. Then, accessing the earliest pattern is $O(1)$ and insertion/deletion is $O(\log |\textsc{Passive}|)$. Similarily, the set of potential patterns is sorted by deactivation times d_P. For any active pattern there was a positive subrelation test, from which we have obtained an observation interval $[b, f]$ and f is used to sort the set of active patterns. Let T denote the complexity of testing a pattern P of dimension k for $P \sqsubseteq W$. Since k is fixed during support estimation, we consider it as a constant. Let us first examine the subroutines in figure 3.20.

```
 1 funct check_pattern(→ P)                          returns true iff P visible
 2   subrelation_check(P, W, π);
 3   observation_interval(dim(P), (b_π(1), b_π(2), ..), (f_π(1), f_π(2), ..), Δt_win, b, f);
 4   if t_act ∈ [b, f]
 5     then append [b, f] at end of O_P; return true
 6      else return false fi
 7 .
 9 proc process_passive(→ P)                  precondition: t_act has reached a_P
10   pop first interval [a_P, d_P] from E_P;                              O(1)
11   remove P from PASSIVE
12   if check_pattern(P) then insert P into ACTIVE
13                       else insert P into POTENTIAL fi
14 .
16 proc process_all_potential(→ s)   precondition: new- oder full-interval event
17   for P ∈ POTENTIAL do
18       if s occurs in P
19         then if check_pattern(P)
20                 then move P from POTENTIAL to ACTIVE; fi
21       fi
22   od
23 .
25 proc process_potential(→ P)               precondition: t_act has reached d_P
26   remove P from POTENTIAL;
27   if supp_min reachable then insert P into PASSIVE fi
28 .
30 proc process_active(→ P)  precondition: t_act has reached end of observation
31   if (t_act = d_P) ∨ (¬check_pattern(P))
32     then remove P from ACTIVE;
33          if supp_min reachable  then insert P into POTENTIAL fi
34   fi
35 .
```

Figure 3.20: Subroutines of support estimation. Input parameter P denotes a temporal pattern, s the label of the interval associated with the new/full-interval event.

Subroutine *check_pattern*: Testing $P \sqsubseteq W$ is $O(T)$. From an instance π the observation interval is calculated in $O(k)$ (Theorem 2). Adding a new observation

interval to the sorted list of intervals is $O(1)$, since any new observation has to be inserted at the end of the list. This gives use the overall complexity $O(T)$.

Subroutine *process_passive*: Since E_P is sorted, popping the first element is $O(1)$. Removing/inserting a pattern from a sorted set S is $O(\log|S|)$, therefore the overall complexity is $O(\log|\textsc{Passive}| + T + \max\{\log|\textsc{Active}|, \log|\textsc{Potential}|\})$. With $|C_k| \geq \max\{|\textsc{Passive}|, |\textsc{Active}|, |\textsc{Potential}|\}$ we have the complexity $O(\log|C_k| + T)$.

Subroutine *process_all_potential*: Due to a New-Interval or Full-Interval event (cases (a) and (b)), potential patterns might become active. There is no way to forsee when this will happen, we therefore have to test *all* potential patterns against W. Since we check potential patterns continuously a newly introduced or fully visible state interval (b, f, s) must be the reason for P to become active. This can be used for a simple pruning test: the algorithm has to be called only in case the state s occurs in P. This test can be done in $O(k)$. The overall complexity is then $O(|\textsc{Potential}| \cdot (T + \log|C_k|))$. (Remark: We can even postpone further checking until s corresponds to the last state in P, as will be shown in section 3.3.3.)

Subroutine *process_potential*: In cases (c) and (d) potential patterns might become passive at their deactivation time. The deactivation time is known in advance and turning potential patterns into passive patterns requires no subpattern test. The online pruning is $O(1)$ since the accumulated length of the continuously updated E_P and O_P can also be updated incrementally. The algorithm complexity is therefore $O(\log|C_k|)$.

Subroutine *process_active*: The complexity is $O(T)$ for the test, $O(\log|\textsc{Passive}|)$ for removing P from Passive, $O(1)$ for the online pruning, and $O(\log|\textsc{Potential}|)$ for the insertion into Potential: $O(T + \log|C_k|)$.

Main routine: The initialization is $O(|C_k|\log|C_k|)$ since all candidates have to be sorted by their activation time in order to build up Passive.

Let us now consider the main loop. Consider the state transitions in figure 3.18 again. How often do the four transitions coming from or leading to node Passive occur? As they are implicitly scheduled by the interval bounds $[a_P, d_P]$ in E_P the total number is $n = \sum_{P \in C_k} \text{card}(E_P)$. Thus, we have n Leave-Passive events and n calls of either *process_active* or *process_potential* with $t_{act} = d_P$ (corresponding to Leave-Active and Leave-Potential events): The total complexity of all these calls is then $O(n \cdot (T + \log|C_k|))$.

The complexity of the actions invoked by the New-Interval and Full-Interval events remains, which correspond to the transitions between the active and potential node in figure 3.18. For each of the L intervals a New-Interval and Full-Interval event is scheduled, which leads to $m = \sum_{i=1}^{L} |\textsc{Potential}_i|$ *process_all_potential* calls: $O(m \cdot (T + \log|C_k|))$ (here $|\textsc{Potential}_i|$ denotes the number of potential patterns in the i^{th} iteration). In the worst case, all potential patterns become active and the observation interval terminates before the deactivation time is reached. Then, $|\textsc{Active}_i| = |\textsc{Potential}_i|$ and *process_active* is called for every potential pattern: $O(m \cdot (T + \log|C_k|))$.

This gives the complexity

$$\begin{aligned} & O(|C_k|\log|C_k| + n(T + \log|C_k|) + m \cdot (T + \log|C_k|)) \\ = \quad & O(|C_k|\log|C_k| + (n+m) \cdot (T + \log|C_k|)) \end{aligned}$$

■

It is difficult to estimate the practical run-time complexity of the support estimation

algorithm, because it depends heavily on the (average) sizes of the three subsets PASSIVE, POTENTIAL, and ACTIVE of C_k. The most expensive operation, however, is checking $P \sqsubseteq W$. In a naive implementation – without partitioning C_k into the three subsets – we would call the subrelation check $4 \cdot L \cdot |C_k|$ times: For each of the L intervals we have four relevant situations (cases (a)-(d)) and in each case we would test every candidate against the sliding window content. Using the partitioning of C_k, we have $n + m$ of these tests. If we understand the transitions from passive to active patterns and vice versa as shortcuts of two consecutive transitions from passive to potential and potential to active, then the number of potential patterns in each iteration increases slightly: The m tests would be performed within the normal processing of potential patterns. Therefore, $\frac{n+m}{L}$ can be considered as the average number of potential patterns during the run. Since we expect from our considerations before (see introducing of the set of passive and active patterns) that the average number of potential patterns will be smaller than the total number of candidates $|C_k|$, we have $n + m < L \cdot |C_k|$ and therefore a reduced complexity compared to a naive implementation.

3.3.3 Continuously Testing for Subpatterns

To estimate the support of a pattern P we have to check continuosly for $P \sqsubseteq W$ while sliding W along the sequence. Can we make use of the fact that the content of the sliding window does not change abruptly but develops slowly over time? Since the subrelation test is expensive, we would like to take advantage from previous tests in order to prune as many unnecessary tests as possible. Consider a certain window position t_{act} and suppose that pattern P is not contained in W, that is, when inspecting the part of the sequence covered by the sliding window we do not observe P. We distinguish two cases:

(a) Consider the case that an instance π of P will be visible shortly and $\pi(\mathbb{N})$ is a subset of the intervals contained in W. This means, that all intervals participating in the instance are already visible, but only partially. As the window slides further, some temporal relationships will be resolved, but we do not need another intervals to eventually observe P. An example for this situation has been given in figure 3.7 with the dashed window position and the pattern "*A contains C*". Although the occurrence of P is not yet decidable from the observers point of view, since we perform analysis on historical data it is possible to forsee the instance from the algorithms point of view.

That is, after some more time, an instance π of P will appear in W, making only use of intervals we already know. Our subpattern-test will find this instance that is about to come into view, so can we already stop checking for P and make the pattern active – although its observation period $[a_P, d_P]$ lies in the future with respect to t_{act}? We would not be allowed to do so, if there is a chance of observing another instance φ of P that starts earlier than π, because then at the time of re-activating P the earlier instance will be disappeared and we will have missed its support. So we have to require from our subpattern-test routine that the *earliest instance* will be returned. (By *earliest instance* we mean the instance whose observation interval starts earliest.)

But can we think of situations where intervals that are still to come can be combined to another pattern instance that occurs even *earlier* than the instance we can forecast right now? This might be possible, because we do not yet know how the content of the sliding window will develop in the meanwhile

(from t_{act} to a_P). However, Theorem 9 in the next section will tell us, that this cannot be the case: An instance φ making use of interval (b_n, f_n, s_n) at position i (that is, $\varphi(i) = n$) cannot start earlier than π: because of $n > \max \pi(\mathbb{N})$, the pointwise minimum of π and φ is π and thus $b_\pi < b_\varphi$. Therefore, once we have found an instance – although it may not be visible right now – we can make the potential pattern active. We only have to consider the observation interval properly.

(b) Secondly, suppose we will never be able to observe an instance π of P using intervals in W only. Whenever an instance φ will be visible in W, we can find the smallest index j such that interval j is not yet in W. Since P is normalized, the instance φ is monotone ($i > j \Rightarrow \pi(i) > \pi(j)$, cf. Theorem 1). This means, that for all $i > j$ we also know that $\varphi(i)$ is also not contained in the current window W. In particular, this is always true for the last state in P, $i = \dim(P)$, regardless of the actual value of j.

Therefore, if we cannot observe an instance of P right now, we may wait until a new state enters the sliding window that matches the symbol of the last state in P. Unless there is no such state, we cannot find an instance of P. This can be used to postpone the next subrelation test once we had a negative test result.

In summary, when continuously checking P for being a subpattern of the sliding window, we can reduce the number of necessary subpattern tests by implementing two modifications: (a) we make use of our knowledge about the interval lengths to find the earliest pattern instances (we do not have to "simulate" the operators view) and (b) we do not have to check continuously for new instances, but only if a new interval enters the sliding window which matches the symbol of the last state.

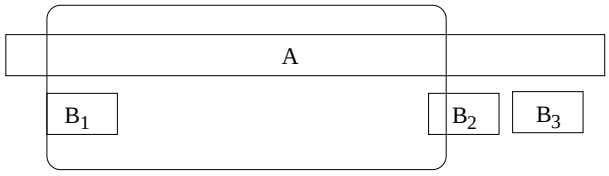


Figure 3.21: At the end of the observation of "A contains B_1" the subpattern test routine must not yield this instance as a valid instance any longer.

In case (a) we make use of the fact the we know future interval bounds even if they are currently not visible in the sliding window. The subpattern test uses these intervals just as if they were visible and we do not loose the correctness, as we have shown above. However, we must not do so with intervals whose bounds lie in the past, especially in combination with modifications according to case b). Figure 3.21 shows an example of a pattern "A contains B", where the sliding window position is drawn where the observation of the A/B_1 pair ends. Thus, the active pattern becomes a potential pattern and we re-check for an occurrence. If the pattern test does not realize that the A/B_1 pair is no longer an instance of the pattern, it will return A/B_1 once more as the earliest instance. The A/B_2 instance will be overlooked. According to modification b) we will not check for the pattern unless B_3 enters the sliding window. Now, A/B_2 will be returned as the earliest instance and A/B_3 is missed. If there is no B_4 following, we loose the support of A/B_3. In a correct implementation, A/B_1 will not be accepted as a pattern instance, because we cannot compare the left bounds of A and B_1 any longer and thus are not sure about the temporal relationship. The earliest (and only) pattern at that point is then A/B_2. At the end of its observation, A/B_3 will be found.

For case (b) the subpattern test algorithm in figure 3.10 has to be changed slightly: Once we reach the last symbol of the pattern during backtracking we do not test all remaining possibilities but only the interval that entered the sliding window most recently.

The modification (a) makes the handling of FULL-INTERVAL events (cf. figure 3.20) obsolete, it can be removed from the algorithm. The intention of these events was to check patterns again as soon as an interval becomes fully visible, that is, its right bound enters the sliding window. According to modification (a) we can handle these cases as soon as the intervals left bound enters the window and thus there is no need for such a special treatment any longer.

What remains at this point is the requirement on our subpattern test to yield the *earliest instance* of a pattern. In the following section we will consider the consequences of this requirement for our subpattern test routines.

3.3.4 Properties of Pattern Instances

Obviously, a pattern P may occur several times within a long sequence. Do we have to enumerate all possible instances to decide which one occurs first? We have seen in the previous section that it is necessary for the correctness of the pruning mechanisms that the pattern matching subroutine always yields the first occurrence of the pattern. Does the algorithm in figure 3.10 or its modified version fulfil this requirement?

Let us assume that a lexicographically ordered sequences $(b_i, f_i, s_i)_{1 \leq i \leq L}$ is given. An instance of a normalized temporal pattern P (subpattern of the sequence) is identified via a state mapping π which associates the i^{th} entry in the pattern P with the $\pi(i)^{th}$ state in the sequence. We will use π and the pattern instance $\{(b_{\pi(i)}, f_{\pi(i)}, s_{\pi(i)}) \,|\, 1 \leq i \leq \dim(P)\}$ synonymously.

Definition 6 *A temporal pattern P is called connected, if the (undirected) graph $G = (V, E)$ is connected, where the set of vertices is given by $V = \{1, .., \dim(P)\}$ and the set of edges is given by $E = \{(i, j) \,|\, R[i, j] \in \mathcal{I}_C\}$ with $\mathcal{I}_C = \mathcal{I} \backslash \{\mathit{after}, \mathit{before}, \mathit{contains}, \mathit{during}\}$.*

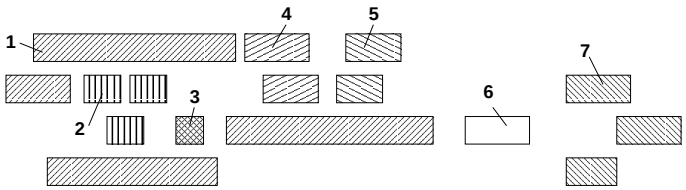


Figure 3.22: Examples of connected temporal patterns.

Figure 3.22 shows some examples of connected subpatterns. State intervals that belong to the same connected pattern have the same texture. With respect to instances of connected patterns we have the following

Theorem 6 *Given a connected temporal pattern P and two instances π, φ of P. Then both instances are identical or do not share any states:*

$$(\exists 1 \leq i \leq \dim(P) : \pi(i) = \varphi(i)) \;\Leftrightarrow\; (\forall 1 \leq i \leq \dim(P) : \pi(i) = \varphi(i)) \qquad (3.16)$$

Proof of Theorem 6: "$\Leftarrow$": obvious. "$\Rightarrow$": Let P be a connected temporal pattern and $G = (V, E)$ the associated graph. Let π, φ be two instances of P with $\pi(k) = \varphi(k)$ for a $k \in \{1, ..., \dim(P)\}$.

Let us assume that the theorems statement is false, that is $\pi(j) \neq \varphi(j)$ for some j. Due to the connectedness of P we can find an edge (i, j) with $\pi(i) = \varphi(i)$ and $\pi(j) \neq \varphi(j)$. Since π and φ match the same pattern P, we have $s_{\pi(j)} = s_{\varphi(j)}$ and from the maximality assumption (3.1) we conclude that $[b_{\pi(j)}, f_{\pi(j)}] \cap [b_{\varphi(j)}, f_{\varphi(j)}] = \emptyset$. More precise, there is a *before* relationship between intervals $\min\{\pi(j), \varphi(j)\}$ and $\max\{\pi(j), \varphi(j)\}$ in the sequence.

Let us now consider the possible cases of $r := R_P[i, j] \in \mathcal{I}_C$: With *is-met-by*, *finishes*, *is-started-by*, *equals*, *starts*, *is-finished-by* and *meets* we can find at least one $b \in \{b_{\pi(i)}, f_{\pi(i)}\}$ that conincides with either $b_{\pi(j)}$ or $f_{\pi(j)}$. For φ to be an instance of the same pattern P, b must also match the corresponding bound $b_{\varphi(j)}$ or $f_{\varphi(j)}$, respectively. Due to the zero intersection of intervals $\varphi(j)$ and $\pi(j)$ this cannot occur. Therefore we have a different interval relationship than required by R_P, which is a contradiction to our assumption.

In case of *is-overlapped-by* and *overlaps* we can find an interval bound $b \in \{b_{\pi(i)}, f_{\pi(i)}\}$ such that $b \in [b_{\pi(j)}, f_{\pi(j)}]$. Again, due to the zero intersection of $[b_{\pi(j)}, f_{\pi(j)}]$ and $[b_{\varphi(j)}, f_{\varphi(j)}]$ we would obtain a different temporal relationship when using $\varphi(j)$ since $b \notin [b_{\varphi(j)}, f_{\varphi(j)}]$. Again, the assumption most be wrong.

The contradictions show that the assumption was false and proves the theorem's statement. ■

Theorem 7 *Given a connected temporal pattern P and two instances π, φ of P. Then the following super-monotonicity property holds:*

$$(\exists 1 \leq i \leq \dim(P) : \pi(i) > \varphi(i)) \Leftrightarrow (\forall 1 \leq i \leq \dim(P) : \pi(i) > \varphi(i)) \qquad (3.17)$$

Proof of Theorem 7: "$\Leftarrow$": obvious. "$\Rightarrow$": Let π, φ be instances of a connected pattern P and $\pi(i) > \varphi(i)$ for some i. Now, let us assume that the theorem's statement is false. Due to the connectedness of P we then find an edge (i, j) in the graph of P such that $\pi(i) > \varphi(i)$ and $\pi(j) \leq \varphi(j)$.

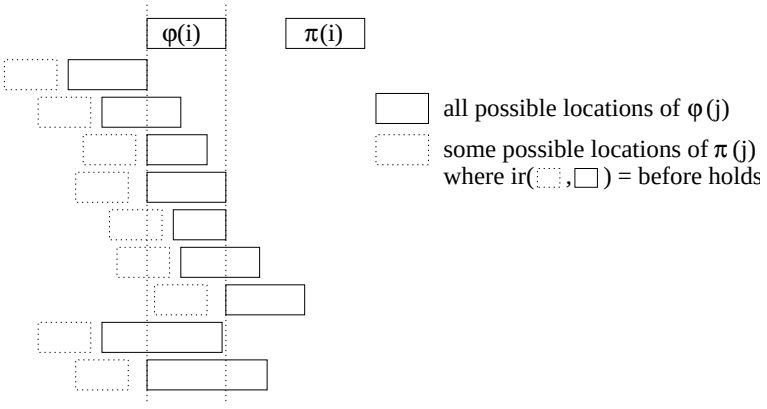


Figure 3.23: Situation in theorem 7.

Figure 3.23 depicts the situation: Due to the maximality assumption and $s_{\pi(i)} = s_{\varphi(i)}$ we know that $\text{ir}([b_{\varphi(i)}, f_{\varphi(i)}], [b_{\pi(i)}, f_{\pi(i)}]) =$ *before*, i.e. both intervals are disjoint and in particular

$$f_\varphi(i) < b_{\pi(i)}. \qquad (3.18)$$

Due to the connectedness of P we find an index j with $R_P[\min(i,j),\max(i,j)] \in \mathcal{I}_C$. In figure 3.23 all possible positions of $[b_{\varphi(j)}, f_{\varphi(j)}]$ are shown. In all cases we have $f_{\varphi(j)} \geq b_{\varphi(i)}$ and

$$b_{\varphi(j)} \leq f_{\varphi(i)}. \tag{3.19}$$

A necessary condition for π being an instance of the same pattern P is therefore

$$(f_{\pi(j)} \geq b_{\pi(i)}) \wedge (b_{\pi(j)} \leq f_{\pi(i)}). \tag{3.20}$$

We do not have to consider the case of $\pi(j) = \varphi(j)$ due to theorem 6 and $\pi(i) > \varphi(i)$. Thus, we consider the case of $\pi(j) \leq \varphi(j)$. From the maximality assumption and $s_{\pi(j)} = s_{\varphi(j)}$ we again obtain $\text{ir}([b_{\pi(j)}, f_{\pi(j)}], [b_{\varphi(j)}, f_{\varphi(j)}]) = \textit{before}$ and in particular $f_{\pi(j)} < b_{\varphi(j)} \leq^{(3.19)} f_{\varphi(i)} <^{(3.18)} b_{\pi(i)}$. This contradicts the left inequality in (3.20) and therefore the assumption was false. ■

We denote the observation interval of an instance π by $[b_\pi, f_\pi]$. A state embedding π can be written as a tuple $(p_1, ..., p_n)$ using $\pi(i) = p_i$. We have already used lexicographical ordering before (theorem 1), but this time the (lexicographically) first pattern yields not necessarily the earliest observation interval. An example sequence is shown in figure 3.24, in which we are looking for the subpattern "(*A contains B*)*before B*". There are four possible solutions $\Pi = \{(1,2,4),(1,2,5),(1,3,4),(1,3,5)\}$. When comparing the solutions $\pi = (1,2,5)$ against $\varphi = (1,3,4)$ we have $\pi < \varphi$ but $b_\pi > b_\varphi$.

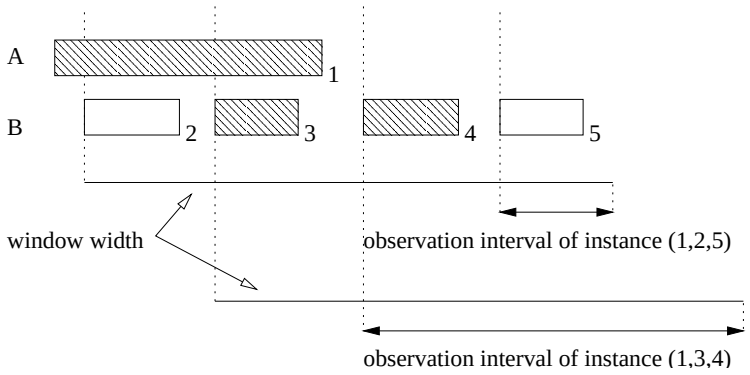


Figure 3.24: Observation interval.

Next, we introduce *before* and *contains* relationships for patterns rather than intervals.

Definition 7 *Given a lexicographically sorted sequence $(b_i, f_i, s_i)_{1\leq i\leq n}$, two patterns $P = (s_P, R_P)$ and $Q = (s_Q, R_Q)$ and disjoint sets I and J such that $(b_i, f_i, s_i)_{i\in I}$ is an instance of P and $(b_j, f_j, s_j)_{j\in J}$ is an instance of Q. We denote the normalized temporal pattern obtained from the sequence $(b_i, f_i, s_i)_{i\in I\cup J}$ by P before Q or P contains$_X$ Q for a set $X \subseteq I$ if*

$$R_{P \textit{ before } Q}[i,j] = \begin{cases} R_P[i,j] & \textit{if } i,j \in I \\ R_Q[i-\dim(P), j-\dim(P)] & \textit{if } i,j \in J \\ \textit{before} & \textit{if } i \in I, j \in J, i < j \end{cases}$$

$$R_{P \textit{ contains}_X Q}[i,j] = \begin{cases} R_P[i,j] & \textit{if } i,j \in I \\ R_Q[i-\dim(P), j-\dim(P)] & \textit{if } i,j \in J \\ \textit{before} & \textit{if } i \notin X, i \in I, j \in J, i < j \\ \textit{contains} & \textit{if } i \in X, j \in J, i < j \end{cases}$$

(In the definition of the interval relationship matrix it is sufficient to deal with the case of $i < j$, then $R[i,i] = equals$ and $R[j,i]$ is the inverse of $R[i,j]$.)

Let us revisit figure 3.22. By P_n we denote the temporal pattern that is defined by all intervals that have the same texture as the interval number n in the figure. This definition generalizes canonically to patterns $P_{n,m}$. Example patterns of the P *before* Q type are "P_4 *before* P_5" and "P_3 *before* P_7". Examples for P *contains*$_X$ Q are given by "P_1 *contains*$_X$ $P_{2,3}$" and "P_1 *contains*$_Y$ $P_{4,5}$", where X contains the indices of the second and third interval in pattern P_1 and Y the index of the fourth interval in P_1.

The next theorem shows that the space of temporal patterns is composed out of connected temporal patterns and the previously defined pattern compositions:

Theorem 8 *Every temporal pattern can be constructed by the following production rules:*

$$P \quad ::= \quad C \mid P \text{ before } P \mid P \text{ contains}_I \ P$$

where C is an arbitrary connected pattern.

Proof of Theorem 8: "$\Leftarrow$": From the definition of the *before* and *contains*$_I$ it is obvious, that these combinations of temporal patterns lead again to temporal patterns. "$\Rightarrow$": We show the statement by induction of the size of P. For $\dim(P) = 1$ we have only connected patterns, the statement is therefore trivially true. Let us now assume that the statement is true for all $\dim(P) < n$. Let P be an arbitrary, normalized temporal pattern of size n. Choose $i \in \{1,..,n\}$ and let I be the set of state indices that belong to the connected subpattern of P that contains state i. If $|I| = \dim(P)$ the whole pattern is connected (production rule $P ::= C$). Let us therefore consider $|I| \neq \dim(P)$. From the definition of connected patterns we then conclude for

$$K := \{R_P[\min(i,j), \max(i,j)] \mid i \in I, j \in \{1,..,\dim(P)\} \backslash I\}$$

that $K \subseteq \{before, contains\}$.

Case (1): $K = \{before\}$, *that is, there is no contains relationship.* This means that any state with index $j \in \{1,..,\dim(P)\}\backslash I$ has a *before* or *after* relationship to any of the states in I. Therefore, the set of indices $\{1,...,\dim(P)\}$ naturally partitions into $\{J_1, I, J_2\}$ with $J_1 = \{j | 1 \leq j < \min I\}$ and $J_2 = \{j | \max I < j \leq \dim(P)\}$ due to the normalization of P (and the maximality assumption). Choose sets K_1, K_2 either as $J_1 \cup I$, J_2 or J_1, $I \cup J_2$ such that both K_1 and K_2 are not empty. Then for the temporal patterns P_1 and P_2 associated with the sets of indices K_1 and K_2 the relationship $P = P_1 \, before \, P_2$ holds (production rule $P ::= P$ *before* P).

Case (2): *There is at least one $i \in I$ and $j \notin I$ such that $R_P[\min(i,j), \max(i,j)] = contains$.* Define X_k for any $k \in \{1,..,\dim(P)\}\backslash I$ to be $\{i' \in I \mid R_P[\min(k,i'), \max(k,i') = contains\}$, that is, X_k is the set of intervals in I that are contained in interval k. We show that X_k is either $\emptyset$ or I by showing for any $k \notin I$:

$$\begin{aligned} & \exists i' \in I : R_P[\min(k,i'), \max(k,i')] = contains \\ \Rightarrow \quad & \forall i' \in I : R_P[\min(k,i'), \max(k,i')] = contains \end{aligned}$$

Let $i' \in I$ with $R_P[\min(k,i'),\max(k,i')] = \textit{contains}$. Assume there is an $j' \in I$ such that $R_P[\min(k,j'),\max(k,j')] \neq \textit{contains}$. Then, since all intervals in I are connected and $k \notin I$, we can conclude that $R_P[\min(k,j'),\max(k,j')] = \textit{before}$. That is, we have an interval i' contained in k and an interval j' outside (before/after) k; choose $t \in \{b_k, f_k\}$ such that t lies between the intervals $[b_{i'}, f_{i'}]$ and $[b_{j'}, f_{j'}]$. Due to the connectedness of I, there must be a path in the graph of I, leading from i' to j'. Due to the non-zero intersections of neighboured intervals in the graph we thus find an interval $[b_{k'}, f_{k'}]$ on this path that contains t. But then the temporal relationship $R_P[\min(k,k'),\max(k,k')] \notin \{\textit{before}, \textit{contains}\}$. This is a contradiction to $k \notin I$. Therefore the assumption was wrong and the statement is shown.

For $X = X_j$ we have $X \neq \emptyset$. With $K_1 = \{1,...,\dim(P)\}\backslash I$ and $K_2 = I$ we have shown that P_{K_1} $\textit{contains}_X P_{K_2}$.

The subpatterns induced by K_1 and K_2 in both cases can be decomposed by induction hypothesis. Therefore, the statement has been shown for $\dim(P) = n$. ■

Finally, we show

Theorem 9 *Given two instances ϕ and ψ of the same pattern P, then $\pi = \min(\phi,\psi)$ $(i \mapsto \min(\phi(i),\psi(i)))$ is also an instance of P and $b_\pi \leq \min(b_\phi, b_\psi)$ holds.*

Proof of Theorem 9: First, we will show that the (pointwise) minimum of two instances yields an instance also. We prove this by induction over the size of the pattern P. At size $\dim(P) = 1$ the minimum of two instances φ and ψ is always either φ or ψ and the statement is trivially true. Let us now assume that the theorem's statement is true for all $\dim(P) < n$ and let us consider the case of $\dim(P) = n$. Let φ and ψ be instances of a pattern P with $\dim(P) = n$. According to theorem 8 we have one of the following three cases:

Case (1): *P is connected.* In this case, due to the super-monotonicity of connected patterns (see theorem 7), π is either φ or ψ and the statement is trivially true.

Case (2): *P can be decomposed into P_1 before P_2.* Considering an instance ϑ as a set of pairs $(i,\vartheta(i))$, we partition φ into φ_1 and φ_2, such that $\varphi = \varphi_1 \cup \varphi_2$, $\varphi_1 \cap \varphi_2 = \emptyset$ and φ_i contains the instances of subpattern P_i. Same for ψ. Since $\dim(P_1) < n$ and $\dim(P_2) < n$ the induction hypothesis holds for φ_1, ψ_1 and φ_2, ψ_2, that is, $\pi_1 := \min(\varphi_1, \psi_1)$ and $\pi_2 = \min(\varphi_2, \psi_2)$ are instances of P_1 and P_2. We have to show that the recombination of π_1 and π_2, namely $\pi = \pi_1 \cup \pi_2$, is also an instance of P. Let us denote the pattern obtained by π as Q.

For any $i,j \in \{1,...,\dim(P)\}$ we have to show that $R_P[i,j] = R_Q[i,j]$. It is sufficient to consider only the cases $i < j$. Obviously, if both i and j belong either to π_1 or π_2 the equality is fulfilled. Let us therefore assume that i belongs to π_1 and j to π_2. Then we have $R_P[i,j] = \textit{before}$ and we have to show $R_Q[i,j] = \textit{before}$, or equivalently $f_{\pi(i)} < b_{\pi(j)}$. From $R_P[i,j] = \textit{before}$ we know that $f_{\varphi_1(i)} = f_{\varphi(i)} < b_{\varphi(j)} = b_{\varphi_2(j)}$ and $f_{\psi_1(i)} = f_{\psi(i)} < b_{\psi(j)} = b_{\psi_2(j)}$. Then also $f_{\pi(i)} = \min(f_{\varphi_1(i)}, f_{\psi_1(i)}) < \min(b_{\varphi_2(j)}, b_{\psi_2(j)}) = b_{\pi(j)}$ holds and therefore $R_Q[i,j] = \textit{before}$.

Case (3): *P can be decomposed into P_1 $\textit{contains}_X$ P_2 for some $X \subseteq \{1,..,\dim(P)\}$.* We partition φ and ψ as in the previous case and denote the pattern of instance

π by Q. For any $i, j \in \{1, ..., \dim(P)\}$, $i < j$, we have to show $R_P[i,j] = R_Q[i,j]$. As before we consider only the case when i belongs to π_1 and j to π_2. We have to show: $R_Q[i,j] =$ *contains* if $i \in X$ and $R_Q[i,j] =$ *before* otherwise.

(a) $i \in X$ and $\varphi(i) = \psi(i)$: Then interval $\pi(i)$ $(= \varphi_1(i) = \psi_1(i))$ contains both, interval $\varphi_2(j)$ and $\psi_2(j)$, and especially $\pi(j) = \min(\varphi_2(j), \psi_2(j))$. Thus $R_Q[i,j] =$ *contains*.

(b) $i \in X$ and $\varphi(i) \neq \psi(i)$: From the maximality assumption we know that the intervals $\varphi_1(i)$ and $\psi_1(i)$ are disjoint. Let us assume $\varphi_1(i) < \psi_1(i)$. Then, from the normalized form, $R_P[i,j] =$ *contains*, and theorem 1 we conclude $\varphi_1(i) < \varphi_1(j) < \psi_1(i) < \psi_2(j)$. Therefore, $\pi(i) = \varphi_1(i)$ and $\pi(j) = \varphi_2(j)$ and thus $R_Q[i,j] =$ *contains*.

(c) $i \notin X$ and $\varphi(i) = \psi(i)$: Then interval $\pi(i)$ is *before* both, interval $\varphi(j)$ and $\psi(j)$, and especially *before* $\pi(j)$.

(d) $i \notin X$ and $\varphi(i) \neq \psi(i)$: From the maximality assumption we know that intervals $\varphi(i)$ and $\psi(i)$ are disjoint. Let us assume $\varphi(i) < \psi(i)$, then we have $f_{\varphi(i)} < b_{\psi(i)}$ and thus $f_{\pi(i)} = f_{\varphi(i)}$. From the *before* relationship in P we know $f_{\varphi(i)} < b_{\varphi(j)}$ and $f_{\psi(i)} < b_{\psi(j)}$, that is $f_{\pi(i)} < \min(b_{\varphi(j)}, b_{\psi(j)})$. (The symmetric case $\varphi(i) > \psi(i)$ can be shown analogously.)

Only $b_\pi \leq \min(b_\varphi, b_\psi)$ remains to be shown. Recall how b_π is obtained: We start to observe the pattern P at $t_{nearmax}$ where t_i are the ordered interval bounds of the instance (cf. page 55). Since the minimum operation leaves the bounds unchanged or decreases them, the value of $t_{nearmax}$ also remains constants or decreases. Therefore $b_\pi \leq \min(b_\varphi, b_\psi)$ and the theorems statement is shown. ■

To find the earliest instance of a given pattern it is therefore necessary to find the instance that is the pointwise minimum of all possible instances. Obviously, algorithm 3.10 fulfils this requirement, if both patterns P and W are in normalized form, because intervals with lower indices are tried first by the algorithm. For the modified algorithm we have to sort the interval pairs lexicographically such that interval pairs with smaller indices are tried first.

3.4 Evaluation

To test the pattern enumeration algorithm we have created three test set types (from which test sets of varying length can be generated). Each type contains a number of simulated variables, where four different labels for each variable have been used. Associated with each variable there is a maximum interval length l and the actual interval length is then uniformly sampled from $\{1, .., l\}$ (cf. figure 3.25). Consecutive intervals meet each other and are labeled randomly. These sequences are merged to the respective test sets. The ratio of the total number of intervals in the sequence and the length of the sequence gives us the average number of intervals per unit length and is denoted by ϱ. In a window of width Δt_{win} we thus observe on average $\Delta t_{win} \cdot \varrho$ intervals.

All experiments were conducted on an AMD 1.4GHz processor with 256 MB RAM under Linux. To our surprise, the simple subpattern test of figure 3.10 outperformed the more sophisticated test discussed on page 58 in all experiments. It seems that even in the simple approach there is not much backtracking, such that the additional organizational overhead cannot be compensated by actively influencing the

	number of variables	total number of labels	used values for l	average density ϱ
type 1	3	12	3, 6, 9	0.98
type 2	6	24	3, 6, 9, 4, 7, 10	1.82
type 3	9	36	3-11	2.54
real	3	8	-	0.13

Figure 3.25: Test set characteristics.

backtracking depth. We assume that the fact that the problematic cases discussed in section 3.2.2 are pathological and occur seldomly in practice.

Figure 3.26 shows the results for data sets of type 2 of increasing size (from top to bottom, approximately 11000, 22000, 33000 intervals). The miminum support was 3%, the window width was 12, yielding an average number of $12\varrho = 21.84$ intervals in the sliding window. For each run and every pattern size k, the number of candidate and frequent patterns (as well as their ratio) is shown. During candidate generation, we perform three pruning tests. The number of patterns that are pruned due to the transitivity check are shown in column p_1, due to the subpattern check in column p_2, and due to the support intersection check in column p_3. The sum of p_1, p_2, p_3 and $|C_k|$ tells us how many patterns of size k have been considered at this stage. The pruning techniques seem to be very efficient, 50% of all candidates are actually frequent patterns, only 1% of the considered patterns became candidate patterns. The last two columns display the run time in seconds for the support estimation (t_{SE}) and candidate generation steps (t_{CG}). The total run time is shown in the last row of each run. As expected, the run-time increases linearly with the data set size. Minor variations may be caused by differences in the randomly generated test sets and the impact on the support sets of the patterns.

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	24	100%	24	0	0	0	0.27	0
2	2085	57.7%	3612	0	0	0	9.22	0.12
3	12183	48.8%	24946	749338	144520	214875	21.36	10.28
4	1399	59.3%	2360	646262	1071521	157887	8.83	17.33
5	0	0%	0	17289	27015	576	0.95	0.13
$\sum$	16404	51.8%	1%	46.4%	40.1%	12.5%	70.63	
1	24	100%	24	0	0	0	0.53	0
2	2116	58.6%	3612	0	0	0	18.88	0.24
3	11910	48.6%	24481	775553	144474	223000	42.36	22.67
4	1445	58.1%	2488	606754	1045674	158925	17.25	37.24
5	0	0%	0	18429	27453	710	1.82	0.23
$\sum$	15495	50.6%	1%	46.3%	39.9%	12.8%	139.17	
1	24	100%	24	0	0	0	0.8	0
2	2117	58.6%	3612	0	0	0	28.46	0.37
3	11746	48.2%	24371	776306	144629	223199	62.74	36.48
4	1414	59.2%	2388	584357	1031493	158562	26.53	58.88
5	0	0%	0	18420	25739	740	2.69	0.32
$\sum$	15301	50.3%	1%	46.1%	39.9%	13%	214.26	

Figure 3.26: Experiments with (from top to bottom) increasing data set size.

Figure 3.27 shows the results for a type 1 data set of constant size (11000 intervals, window width 16). This time the support threshold is varied (from top to bottom

2%, 1%, $\frac{1}{2}$%). The support threshold is of vital importance for the efficiency of the algorithm, when it is set to zero we enumerate the complete pattern space (and we have illustrated its tremendous size in section 3.3.1). From the figure we can see that we have more frequent 5-patterns when using a support threshold of $\frac{1}{2}$% than candidates in the whole run for 2%. The percentage of frequent patterns among the candidates increases up to 93%, almost every pattern becomes frequent. Consequently, the run-time increases drastically.

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	12	100%	12	0	0	0	0.14	0
2	554	63.7%	870	0	0	0	1.25	0.02
3	9367	77.8%	12047	104774	17043	18959	7.26	1.1
4	13111	77.4%	16950	820731	444700	181277	13.18	11.58
5	2351	91.4%	2572	305591	510617	67638	6.81	7.52
6	0	0%	0	20893	20206	525	0.89	0.13
$\sum$	25395	78.2%	1.3%	49.2%	38.8%	10.7%	48.86	
1	12	100%	12	0	0	0	0.14	0
2	626	72%	870	0	0	0	1.08	0.02
3	14154	78.6%	18000	139496	16726	20563	7.95	1.25
4	47985	87.9%	54567	1656762	639327	339130	23.79	17.48
5	14662	93.5%	15682	2095675	1876600	281977	17.46	25.39
6	109	99.1%	110	217159	401338	19292	3.91	2.75
7	0	0%	0	143	70	0	0.19	0
$\sum$	77548	86.8%	1.1%	52.7%	37.6%	8.6%	101.22	
1	12	100%	12	0	0	0	0.14	0
2	664	76.3%	870	0	0	0	1.08	0.02
3	19921	81%	24585	159313	16093	18707	8.61	1.32
4	132688	94.5%	140392	3043123	833719	533734	38.19	23.62
5	62664	95.8%	65408	8011113	5678624	1012778	41.01	69.37
6	5745	99.9%	5753	1368390	1823874	81586	12.82	13.14
7	14	100%	14	38773	38669	309	1.16	0.22
8	0	0%	0	14	0	0	1.77	0
$\sum$	221708	93.5%	1%	55.1%	36.6%	7.3%	212.47	

Figure 3.27: Experiments with (from top to bottom) decreasing minimum support threshold (2%, 1%, $\frac{1}{2}$%).

The number of frequent patterns also increases if the window width is increased. On the one hand, new pattern instances may become observable if the window is enlarged, on the other hand, visible patterns can be observed for a longer period of time. In general, the support values will increase and therefore the number of candidates and frequent patterns (if the minimum support threshold is constant). Figure 3.28 shows the results of some experiments where the window width has been increased from 8 to 12 and 16. Data set type 3 has been used (15000 intervals, 3% minimum support), we thus obtain an average number of intervals in the sliding window of 20.32, 30.48, and 40.64, resp. As expected, the number of candidate and frequent patterns increases quickly and so does the runtime. For several values of k the time needed to generate and prune candidates is even larger than the time required for support estimation.

When increasing the window width or decreasing the minimum support we quickly run into situations where the run time is no longer manageable. However, this is not due to some deficiencies of the algorithm, but caused by high number of frequent patterns found. If we consider the average run time per 100 discovered frequent patterns we obtain 0.19s, 0.13s, 0.10s in case of figure 3.27 and 1.20s, 0.61s, 0.51s

in case of figure 3.28. Thus, the efficiency of the algorithm is even increasing – but outweighed by the increase in the number of frequent patterns. For very large window widths or very small minimum support the assumptions discussed at the beginning of section 3.3.1 do no longer hold. (A potential solution for increasing window width will be proposed in section 5.3).

The increasing window width has also some (neglectable) positive effects, since the average number of intervals in the support sets O_P/E_P decreases (the probability that the support of neighbouring instances overlap increases). During candidate generation we intersect these sets and since the complexity depends on the number of intervals, the intersection is done more quickly. This can be observed for $k = 2$ in figure 3.28, because at that stage the sets of frequent 1-patterns are identical in all runs. The elapsed time during candidate generation reduces from 0.39s to 0.2s.

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	36	100%	36	0	0	0	0.38	0
2	3573	43.4%	8226	0	0	0	25.8	0.39
3	1377	15.5%	8910	1370101	452976	438335	8.13	23.22
4	0	0%	4	24693	88490	1420	1.31	0.73
$\sum$	4986	29%	0.7%	58.3%	22.6%	18.4%	59.96	
1	36	100%	36	0	0	0	0.37	0
2	4614	56.1%	8226	0	0	0	32.63	0.28
3	26757	41.9%	63908	2455229	491980	743737	63.33	34.58
4	1634	54.8%	2983	1652590	3461619	403607	17.93	51.34
5	0	0%	0	20304	41497	580	1.19	0.19
$\sum$	33041	43.9%	0.8%	44.2%	42.7%	12.3%	201.84	
1	36	100%	36	0	0	0	0.37	0
2	5115	62.2%	8226	0	0	0	38.23	0.2
3	108405	62.2%	174149	3104590	489901	839904	181.27	37.26
4	68718	51.4%	133776	17655786	16321661	3763397	178.75	388.63
5	133	44%	302	1483574	6537951	373461	38.71	67.55
6	0	0%	0	230	189	0	0.53	0
$\sum$	182407	57.6%	0.6%	43.7%	45.9%	9.8%	931.5	

Figure 3.28: Experiments with (from top to bottom) increasing window width (8, 12, 16).

Finally, we want to examine the influence of the number of symbols. We choose a window width of 8, 11, and 21 for a data set of type 1, 2, and 3, resp. The average number of intervals in the sliding window is then quite similar for all three cases: 20.58, 20.02, and 20.32, resp. Figure 3.29 shows some results. It can be seen that the number of candidates (and thus the run time) decreases as the number of symbols increases. The percentage of frequent patterns among the candidate patterns also decreases, which is the reason why the average run-time per 100 frequent patterns increases from 0.22s to 0.48s and 1.20s.

To conclude, the experiments have shown that the algorithm is efficient enough to cope with qualitative descriptions of time series. Once potentially interesting relationships have been discovered, the further analysis may concentrate on those time series that participate in the discovered relationship. The reduction in the number of variables can then be used to characterize the remaining series in greater detail (increase in the number of symbols).

As a real world data set we have extracted from 3 time series (air pressure, wind strength, wind direction) an 8 symbol description (increasing/decreasing trend and convex/concave trend for air pressure). Approximately 7.5 years of hourly obser-

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	12	100%	12	0	0	0	0.14	0
2	537	61.7%	870	0	0	0	1.09	0.01
3	10311	78%	13223	97505	17044	16094	9.5	0.82
4	36386	81.5%	44657	953758	496453	210233	38.34	16.42
5	25627	84.9%	30175	1242417	1572211	256642	37.59	37.25
6	7	31.8%	22	382692	817600	83156	11.27	11.36
7	0	0%	0	12	8	0	0.17	0
$\sum$	72880	81.9%	1.4%	42.9%	46.6%	9.1%	163.96	
1	24	100%	24	0	0	0	0.27	0
2	2034	56.3%	3612	0	0	0	8.67	0.13
3	7479	41.8%	17904	708357	143575	208964	14.75	9.87
4	453	61.1%	741	288636	617179	77922	5.32	8.62
5	0	0%	0	4233	5177	70	0.51	0.02
$\sum$	9990	44.8%	1.1%	48.2%	36.9%	13.8%	48.16	
1	36	100%	36	0	0	0	0.38	0
2	3573	43.4%	8226	0	0	0	25.8	0.39
3	1377	15.5%	8910	1370101	452976	438335	8.13	23.22
4	0	0%	4	24693	88490	1420	1.31	0.73
$\sum$	4986	29%	0.7%	58.3%	22.6%	18.4%	59.96	

Figure 3.29: Experiments with (from top to bottom) increasing number of symbols (12, 24, 36) at constant average number of intervals in the window.

vations are thereby condensed to 8585 labeled intervals. Using a window width of three days ($\Delta t_{win} = 72$) and a support threshold of 1%, a total number of 19070 frequent patterns were discovered in 50 seconds (see figure 3.30).

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	8	100%	8	0	0	0	0.29	0
2	210	56.5%	372	0	0	0	1.34	0.01
3	3127	74.2%	4213	20425	4184	3052	6.58	0.83
4	9561	86.8%	11010	223348	80857	38637	12.97	6.79
5	5343	99.3%	5378	311395	189599	11434	11.64	3.61
6	792	100%	792	82463	48766	187	4.21	0.52
7	29	100%	29	5827	2402	1	0.87	0.03
8	0	0%	0	154	44	0	0.43	0
$\sum$	19070	87.4%	2.1%	61.6%	31.2%	5.1%	50.12	

Figure 3.30: Application to weather data.

Chapter 4

Rule Evaluation and Specialization

After having determined all frequent temporal patterns, our next goal is to find informative rules with temporal patterns in the premise and conclusion. For the sake of simplicity we restrict ourselves to "forward rules", that is, rules that make conclusions in the future rather than in the past, but this restriction is not mandatory. Enumeration of all possible rules can be done efficiently using techniques described in [Agrawal et al., 1996]. The number of generated rules is, as well as the number frequent patterns, usually very large. Traditionally, the rule probability p is used to rank the rules according to their confidence. We will show, however, that in case of temporal patterns in interval sequences this measure is not very meaningful. We revise the rule semantics and propose a modification to overcome these difficulties, which are specific to interval patterns. But still, as it has been observed by many authors, confidence alone is not a good indicator for interesting rules. We propose to use the information-theoretic J-measure, which has excellent properties but is also seldomly used in the data mining community (section 4.1). Furthermore, we consider the problem of rule specialization in section 4.2, that is, how to increase the interestingness of rules by adding quantitative constraints on additional attributes of the labeled intervals. The length of the intervals may serve as such an attribute, but also the slope of the represented time series segment or dosage of an admixture in a chemical process. We propose a new algorithm to identify such thresholds automatically from data. This approach is quite unique, since in the literature such thresholds usually address single values in the series (at some specific point in time, e.g. [Kadous, 1999, Rodriguez and Alonso, 2002]) rather than the duration or length of certain observations in the series. In approaches that use symbolic abstractions, these thresholds are usually fixed a priori (e.g. [Carrault et al., 2002]). We will demonstrate the usefulness of the new algorithm in section 4.3.

4.1 Finding Informative Rules

Traditionally, the confidence of a rule $\text{conf}(A \rightarrow B) = \frac{\text{supp}(B)}{\text{supp}(A)}$ has been used to rank the rules. A lower bound on the confidence is usually used to prune the large space of rules. However, it has been observed by many authors, that this measure is not very good in highlighting interesting rules. And this is especially true for temporal rules, as we will see in this section. Let us consider the case when two

patterns perfectly correlate in a labeled interval sequence. Using again the example in figure 3.3, let us assume that whenever we observe "A before B", we find another two intervals "A meets B" afterwards. Usually, the support and confidence value of the rule are used to decide about its usefulness [Agrawal et al., 1996]. If a sequence consists of rule patterns only (cf. figure 3.3), we should expect a confidence value near 1, however, this is not necessarily the case.

Figure 4.1 illustrates this: On the left side we see the premise pattern, on the right side the full rule pattern. In the top row we have the sliding window position at which we just start to see the respective pattern, and in the bottom row we have the position when we loose the pattern. The time that has passed in the meanwhile is the support of the pattern (also depicted in the figure). We can see that the support of the premise pattern is much larger than that of the rule pattern because the extension of the latter is larger. The greater the (temporal) extent of the pattern, the smaller the probability of observing the pattern in the sliding window. Consequently, the confidence of a rule decreases as the extent of the rule pattern increases.

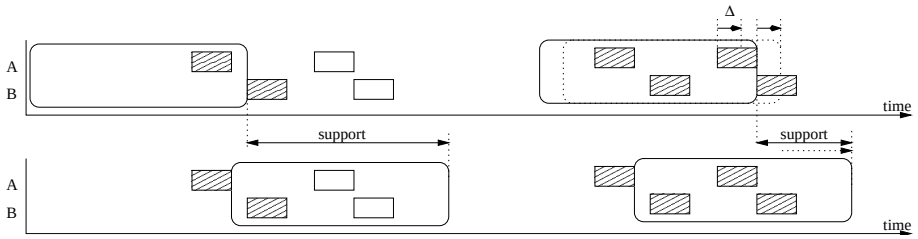


Figure 4.1: Support of patterns decreases with their extent.

4.1.1 Modified Rule Semantics

Thus there are two possible reasons for a low rule probability (or confidence). Firstly, the confidence of a rule decreases as the extent of the rule pattern increases. Secondly, if there are more premise patterns and less rule patterns, rule confidence also decreases. The latter is what we usually associate with rule confidence, whereas the former seems a bit counterintuitive in this context. To reduce the effect of pattern extension, we define a different rule semantics: *Given a randomly selected sliding window that contains an instance of the premise pattern, then with probability p this window overlaps a sliding window that contains the rule pattern.* Loosely speaking, the effect of this redefinition is an increase in the support of the rule pattern, since we substitute "number of windows that contain rule pattern" by "number of windows that contain the premise and overlap a window with a rule pattern".

Figure 4.2(a) illustrates the problem once more. We consider the premise pattern $P =$"A", the conclusion pattern $C =$"B", and the rule pattern $R =$"A before B", w denotes the window width. For any pattern Q, let S_Q be the support set of Q, that is $\text{supp}(Q) = \text{len}(S_Q)$ (cf. definition 5). In the example we have $\text{supp}(P) = \text{len}([a_l, a_r + w])$ and $\text{supp}(C) = \text{len}([b_l, b_r + w])$. Here $\text{supp}(R) = \text{supp}(P \cap C)$ holds and hence $\text{supp}(R) = \text{len}(S_A \cap S_B) = \text{len}([b_l, a_r + w])$. Thus, assuming $\Delta := b_l - a_r < w$ and denoting the length of A by l_A, the rule confidence is

$$\text{conf}(A \rightarrow A \text{ before } B) = \frac{\text{len}(S_A \cap S_B)}{\text{len}(S_A)} = \frac{a_r + w - b_l}{a_r + w - a_l} = \frac{w - \Delta}{w + l_A}$$

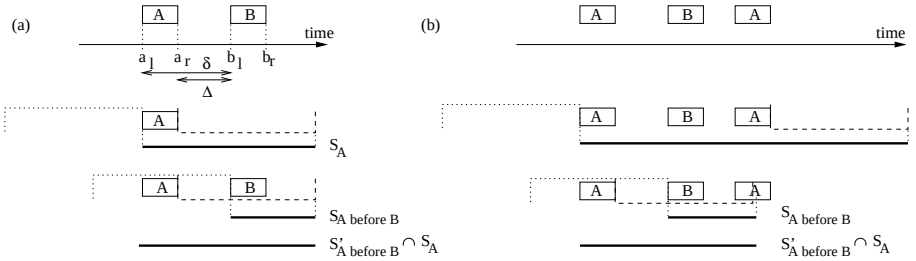


Figure 4.2: Support sets of "A" and "A before B", determined by the sliding window positions when the pattern is observed for the first time (dotted window position) and for the last time (dashed window position).

Obviously, as the gap (Δ) between A and B increases, the confidence approaches zero. Now for any pattern Q, let $S'_Q := S_Q \cup \{t - w \,|\, t \in S_Q\}$. S'_Q can be interpreted as the support of a pseudo-pattern "pattern Q is visible or will be visible within time w". If we now replace the cardinality of "windows that contain rule patterns" by the cardinality of "windows that contain premise and overlap a window that contains rule patterns" as required by the new semantics, we obtain[1]

$$\text{conf}(A \rightarrow A \text{ before } B) = \frac{\text{len}(S_A \cap S'_B)}{\text{len}(S_A)} = \frac{a_r + w - a_l}{a_r + w - a_l} = 1$$

Thus, as long as we can see A and the beginning of B within the sliding window, we obtain a confidence value of 1, no matter how far A and B are apart. Cases where no conclusion pattern occurs are not affected by this modification (see figure 4.2(b)). Thus it helps to recover the usual semantics of confidence values.

The sets S_P have been determined while searching for frequent patterns anyway, they can be handled easily as sorted lists of intervals. Therefore the operations discussed above can be implemented efficiently without looking at the data again: We replace every interval $[l, r] \in S_Q$ by $[l - w, r]$ to obtain S'_Q. In general, rule confidence is then given by

$$\text{conf}(P \rightarrow R) = \frac{\text{len}(S_P \cap S'_R)}{\text{len}(S_P)}$$

4.1.2 Information Content of a Rule

Usually one obtains a large number of frequent patterns and thus a large number of rules. Considerable efforts have been undertaken in the literature to make the vast amount of rules more amenable. Using confidence as a rule measure does not necessarily identify interesting rules: We obtain a confidence of 1 even if premise and rule pattern occur only once (and the minimum support has been chosen that small). Besides that, a rule with high confidence may be quite uninteresting if the probability of observing the conclusion pattern is high anyway. And what about rules with identical confidence values? They are clearly not equally interesting, for instance, the more frequent the premise pattern the more often the rule can be applied. In order to *rank* rules, a rule evaluation measure has to consider all these aspects and provide a single number denoting its interestingness.

We use the J-measure to rank the rules by their information content. It is considered as one of the most promising measures for rule evaluation

[1]Here we have $S_R = S_A \cap S_B$ and therefore $S'_R \cap S_P = S'_A \cap S'_B \cap S_A = S'_B \cap S_A$.

[Berthold and Hand, 1999]. The advantage of the J-measure is that it seems to balance the goodness-of-fit (confidence) and simplicity (support) of the rule very well [Smyth and Goodman, 1991].

Given a rule "if $Y = y$ then $X = x$" on random variables X and Y, the J-measure compares the a priori distribution of X with the a posteriori distribution of X given that $Y = y$. In the context of a rule, we are only interested in two cases, given that $Y = y$, either the rule was right ($X = x$) or not ($X = \bar{x}$), that is, we only consider the distribution of X over $\{x, \bar{x}\}$. Then, the relative information

$$j(X|Y = y) = \sum_{z \in \{x, \bar{x}\}} Pr(X = z|Y = y) \log_2 \left(\frac{Pr(X = z|Y = y)}{Pr(X = z)} \right)$$

yields the *instantaneous* information that $Y = y$ provides about X (j is also known as the Kullbach-Leibler distance or cross-entropy). When applying the rule multiple times, on average we have the information $J(X|Y = y) = Pr(Y = y) \cdot j(X|Y = y)$. The value of J is bounded by ≈ 0.53 bit [Smyth and Goodman, 1991].

In our context, the random variable Y indicates whether the premise occurred in the sliding window W or not. The probability $Pr(P \sqsubseteq W)$ when choosing a sliding window position at random is $\text{supp}(y)/(T + \Delta t_{win})$ where T is the length of the whole sequence. The random variable X indicates whether the rule pattern has occurred. The a priori probability for $R \sqsubseteq W$ is $\text{supp}(S_R)/(T + \Delta t_{win})$, the a posteriori probability is given by $\text{supp}(S_R)/\text{supp}(S_P) = \text{conf}(P \rightarrow R)$. When using the modified rule semantics, we have to replace S_R by $S'_R \cap S_P$.

4.1.3 From Rules to Correlations

Usually premise and conclusion terms in a rule have no obvious correlation. In the case of our temporal rules, however, the probability of observing the rule pattern without observing the premise pattern is zero. This is due to the fact that the rule pattern contains the premise pattern. Although we have defined a conclusion pattern (section 3.1.1), which is determined uniquely by premise and rule pattern, its probability of occurrence does not affect the J-value. The J-measure reflects the goodness-of-fit between $Pr(X)$ and $Pr(X|Y)$, which includes the probability of the conclusion alone in conventional rules, but the probability of the rule pattern in our case.

We can easily consider the relevance of the conclusion C in a rule $P \rightarrow R$ by investigating the reversed rule $C \rightarrow R$. If $P \rightarrow R$ and $C \rightarrow R$ hold, then we have a correlation or equivalence $P \leftrightarrow_R C$, that is, the premise is an indication for the conclusion and vice versa. We simply have to evaluate the J-measure once more, this time Y denotes the random variable that indicates whether the conclusion has been found in the sliding window (or in a window overlapped by it, if we use modified rule semantics), thus $Pr(C \sqsubseteq W) = \text{len}(S_C)/T$. The random variable X is left unchanged. We obtain two J-values for $P \rightarrow R$ and $C \rightarrow R$ and use the sum of both to rate the rules. In a graphical representation we can print the rule $P \rightarrow R$ or $C \rightarrow R$ if one of the J-values is much higher than the other, and $P \leftrightarrow_R C$ if both values are comparable.

4.1.4 Disjunctive Combination of Temporal Patterns

When analysing the rules obtained by the algorithm, we must keep in mind that we were seeking for the simple interval relationships only, that is, those relationships

that consist of a single attribute $r \in \mathcal{I}$. If a process B is started some time after A has started, then this can result in a number of rules "$A \rightarrow B$" with temporal relationships *overlaps*, *meets*, and *before*. The confidence of the *true* relationship (which is in this case: A *overlaps/meets/before* B) might be very high, but the confidence values we observe for the three rules we have found are comparatively low. We are not allowed to add up the confidence values of all three rules in order to obtain the confidence of the composed rule. This would lead to an overestimation, because there might be sliding windows that contain multiple of these patterns simultaneously, and in this case we would count them twice (or more). Fortunately, it is possible to calculate the support of composed rules afterwards.

The support of a pattern P which is a disjunction of two patterns Q and R can be calculated easily as $\text{supp}(P) = \text{card}(O_Q \cup O_R)$. The sets of observed support O_Q and O_R have been calculated already during the execution of the algorithm, all we have to do is to store the sets for later access. (Note that we cannot guarantee that we will find all frequent pattern compositions in this way. Several patterns that do not reach supp_{min} individually might fulfil this requirement after their combination.)

4.2 Quantitative Rule Specialisation

So far, rule evaluation considers the interval relationships of temporal patterns only, but often there is additional information available for each interval. For example, we have not yet evaluated the length of the intervals, or the size of a gap between two intervals, etc. These lengths are always available when dealing with interval data, but there might be additional information attached. For instance, if the intervals denotes ingredients in a chemical process, an additional attribute might denote the intensity or dose of the admixture. A rule that seems interesting to an expert might not have reached the desired confidence value or information content, unless this additional information is incorporated into the rule. For instance, the desired product quality might be achieved only if admixture D has been supplemented to the process at a dose greater than x. In this section we consider the problem of improving the rule quality using such additional label information.

In chapter 3, where our notion of support has been defined, we made use of the fact that testing for a pattern occurrence rather than enumerating all occurrences is sufficient. Does this hold for rule specialization, too? Let us examine the following example:

$$\text{if}\ \underbrace{\begin{array}{c|cc} & A & B \\ \hline A & = & b \\ B & a & = \end{array}}_{\boxed{\texttt{A}}\ \boxed{\texttt{B}}}\ \text{then}\ \underbrace{\begin{array}{c|ccc} & A & B & C \\ \hline A & = & b & b \\ B & a & = & b \\ C & a & a & = \end{array}}_{\boxed{\texttt{A}}\ \boxed{\texttt{B}}\ \boxed{\texttt{C}}}\ \text{with probability } p \tag{4.1}$$

Consider the sequence in figure 4.3, where subfigure (b) shows hypothetical pattern instances as they may be identified during support estimation. Now, let us assume that the "true relationship" requires an additional constraint on the length of B (greater than a certain threshold, fulfilled by intervals 5 and 12). This makes sense in figure 4.3 since there is a long B (interval 5) in the first block (intervals 1-7), no long B for the block in the middle (intervals 8-9) and thus no C interval in the vicinity, but we have a long B (interval 12) in the third block (10-15) and yet another rule pattern. We would expect a rule inducer to find this relationship for the data shown in the figure. However, the depicted interval assignment as returned

by our hypothetic support estimation algorithm by chance does not use intervals 5 and 12 and thus will be unable to discover this condition. Therefore, it is important not to ignore the ambiguity in embedding patterns in state sequences.

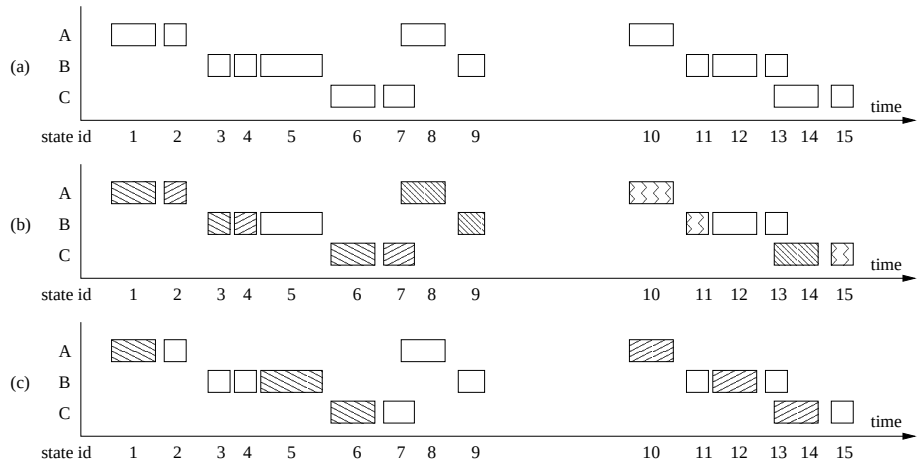


Figure 4.3: Counting the occurrences of temporal patterns. The intervals are enumerated on the time axis, the labels are given by the interval's location on the vertical axis.

Given a rule $P \rightarrow R$ with temporal patterns P and R and a real-valued attribute a attached to one of the labels used in R. We do not make a distinction between attributes of intervals that occur in the premise (e.g. "if $A \wedge \text{length}(A) < 3$ then A before B") or in the conclusion (e.g. "if A then A before $B \wedge \text{length}(B) > 1$"). Potentially it is possible to improve the information content of a rule in both cases (when using the sum of information contents $P \rightarrow R$ and $C \rightarrow R$ as discussed in section 4.1.3). To refine the rule by imposing a constraint on a, we run once through the database and collect all instances of the premise in a set $\mathcal{P}$ and all instances of the rule pattern in a set $\mathcal{R}$ (as well as $\mathcal{C}$ in case of $C \rightarrow R$). For each instance, we store a pair (a, I) consisting of the value a of the selected attribute and the observation interval I (contributing to the support) in the respective set. In contrast to the frequent pattern mining process described in section 3.2.2, now we are not satisfied if we know that there is an occurrence of the rule pattern in the sliding window, but this time we are interested in collecting *all* instances. Since this database pass is computationally more expensive, only selected rules should be considered (e.g., best 100 rules found so far).

Next, we have to find a threshold α such that the J-value of either $P \wedge (a > \alpha) \rightarrow R$ or $P \wedge (a < \alpha) \rightarrow R$ is maximized. Figure 4.4 shows the specialization algorithm. We sort the sets $\mathcal{P}$ and $\mathcal{R}$ by the value of the selected attribute and determine its range $[a_{min}, a_{max}]$. Next, we sweep α once through the set of a values we have found. We start from $\alpha = a_{max}$ to determine the threshold for $a \geq \alpha$ and from a_{min} to determine the threshold for $a \leq \alpha$ (figure 4.4 shows only one case). Whenever a changes its value we consider $\alpha = \frac{a_i + a_{i+1}}{2}$. For every new value of α we incrementally update the support set S_P and S_R for premise and rule pattern, that is, S_P becomes

$$S_P \cup \bigcup_{(a,I) \in \mathcal{P}, a = \alpha} I.$$

in the next step. Given the updated support sets S_P and S_R we can now calculate the J-value for this α. If we want to check for correlations rather than just rules, we additionally maintain the support S_C of the conclusion pattern C. When J

becomes maximal, we have found the α value that yields the most informative rule. Having done this for all available attributes, we specialise the rule with the most informative attribute in a greedy manner. We may refine the specialised rule again, or use the bounds on the J-value [Smyth and Goodman, 1991] to stop when no improvement is possible. The algorithm in figure 4.4 is executed for every attribute of any of the intervals participating in the pattern. If a is not an attribute of the premise pattern P, then S_P is initialized to O_P and not further modified during the algorithm.

---

1 let T be the sequence length plus window width;
2 sort $\mathcal{P}, \mathcal{C}, \mathcal{R}$ by a values;
3 determine range of a values $[a_{min}, a_{max}]$;
4 $\alpha' = \alpha = a_{min}$; $S_P = S_R = \emptyset$; $i_P = i_R = 0$;
5 $\alpha_{best} = -\infty$; $J_{best} = 0$;
6 **while** $\alpha > a_{max}$ **do**
7 $\quad A_P = A_R = \emptyset$;
8 $\quad$ **while** $(|\mathcal{P}| > i_P) \wedge \mathcal{P}[i_P] = (\alpha, I)$ **do** $S_P = S_P \cup I$; $++i_P$; **od**
9 $\quad$ **while** $(|\mathcal{R}| > i_R) \wedge \mathcal{R}[i_R] = (\alpha, I)$ **do** $S_R = S_R \cup I$; $++i_R$; **od**
10 $\quad S_P = S_P \cup A_P$; $S_R = S_R \cup A_R$;
11 $\quad$ calculate J-value (using $Pr(P \sqsubseteq W) = \text{len}(S_P)/T$ etc.)
12 $\quad$ **if** $J > J_{best}$ **then** $J_{best} = J$; $\alpha_{best} = (\alpha' + \alpha)/2$; **fi**
13 $\quad \alpha' = \alpha$; $\alpha = \max\{\mathcal{P}[i_P], \mathcal{C}[i_C], \mathcal{R}[i_R]\}$
14 **od**
15 output: best specialization of $a \geq \alpha$ is $\alpha = \alpha_{best}$

Figure 4.4: Specializing Rules. It is assumed that a specialization variable a has been selected and that the sets $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{R}$ contain pair of a values and support intervals, as described in the text. The ith element of $\mathcal{P}$ is denoted by $\mathcal{P}[i]$.

Theorem 10 *Let $N = \max\{|\mathcal{P}|, |\mathcal{R}|\}$ be the number of instances of premise and rule patterns that we have found. Let m denote the number of distinct values for attribute a and let n_i be the number of occurrences of the ith value of a, that is, $\sum_{i=1}^{m} n_i = N$. Then, the time complexity of the algorithm in figure 4.4 is $O(N \log N + Nm)$.*

Proof of Theorem 10: Sorting the instances by their a value in line 2 is $O(N \log N)$. The while-loop in line 6 is executed m times.

We assume that A_P and A_R are implemented as binary trees to quickly access the closest interval in A given a new interval I. Then, inserting an element is $O(c \log |A|)$, where additional complexity c is needed if I overlaps several intervals in S such that multiple elements in S have to be removed. Since the intervals I are limited in their length (due to the width of the sliding window) in a typical application only a limited number of intervals will be covered by I such that c can be considered as bounded. Thus, for each of the inner while-loops we have $O(\sum_{j=1}^{n_i} c \log j) = O(n_i c \log(N))$.

For the evaluation of the J-measure we have to calculate the union $A_P \cup S_P$, which is linear in $|A_P|$ and $|S_P|$ since both sets are already sorted. The same is true for the calculation of S'_R and intersection of $S'_R \cap S_P$. Thus, we have $O\left(\sum_{j=1}^{i} n_i\right)$.

All together, we obtain

$$\begin{aligned} & O\left(N\log N + \sum_{i=1}^{m}\left(n_i c\log(N) + \sum_{j=1}^{i} n_j\right)\right) \\ = \; & O\Big(N\log N + c\log(N)\underbrace{\sum_{i=1}^{m} n_i}_{\leq N} + \sum_{i=1}^{m}\underbrace{\sum_{j=1}^{i} n_j}_{\leq N}\Big) \\ = \; & O(N\log N + Nm) \end{aligned}$$

■

Apparently, if $N = m$, that is all values of a are distinct, we have $O(N^2)$. It therefore might be advisable to discretize the range of a continuous variable appropriately (e.g. restrict $\Delta\alpha$ to be ≥ 0.01). As we have emphasized before, all possible instances will be counted for N, that is, for the pattern "A before B" and 3 intervals for each label we have up to 3^2 instances of the pattern. But at least m is not affected in such cases, since we have only 3 intervals with the same label, 3 out of 9 pattern instances must have the same value for a.

4.3 Evaluation

In this section we demonstrate the proposed methodology on an artificially generated sequence and an interval sequence that has been extracted from real-world time series. We have restricted ourselves to "forward rules" in the rule enumeration step, that is, rules that make conclusions in the future rather than in the past.

Artificial Test Data

We have generated a test data set where we have randomly switched three variables A, B, and C on and off at discrete time points in $\{1, 2, ..., 9000\}$ with probability 0.2, yielding a sequence with 2838 intervals. Whenever we have encountered a situation where only A is active during the sequence generation, we generate with probability 0.3 a 4-pattern A meets B, B before C, and C overlaps a second B instance. The length and gaps in the pattern were chosen randomly out of $\{1, 2, 3\}$. We have executed the pattern discovery ($\text{supp}_{min} = 2\%$) and rule generation process several times, using the old and new rule semantics and different sliding window widths (8,10,12). We consider the artificially embedded pattern and any subpattern consisting of at least 3 intervals as interesting. As expected, using the old rule semantics the confidence value is not very helpful in finding interesting rules. Most of the top-ranking rules were not interesting. Among the top 10 rules, we have found 1/2/3 interesting patterns for $w = 8/10/12$, they all had 2-3 intervals in the premise and 1 in the conclusion pattern. The J-measure yields much better results, even when using the old semantics. When using the modified semantics, we obtain higher confidence values and J-values. The top few rules rated by J-values were identical, regardless of the window width, among them all 3 possible rules with 4 intervals: $A \leftarrow BCB$, $AB \leftrightarrow CB$, $ABC \rightarrow B$ (arrow direction as indicated by the J-values).

In a second dataset, we have created the described pattern whenever the length of the A interval is greater or equal to 5. For this dataset the rule "$AB \leftrightarrow CB$"

obtained $J_{AB\rightarrow CB} = 0.26$ bit ($\text{conf}_{AB\rightarrow CB} = 75\%$) and $J_{AB\leftarrow CB} = 0.19$ bit ($J = J_{AB\rightarrow CB} + J_{AB\leftarrow CB} = 0.45$ bit). We have searched for a threshold α to specialize the rule. When comparing the rules with different α values as it is done in the algorithm in figure 4.4 we obtain a single rule where J becomes maximal with the correct value ($\alpha = 5$). The confidence increases to 0.85 and the information content by 0.1 bit to $J = 0.55$ bit. In contrast, the confidence value for $\alpha = 5$ represents only a local maximum, beyond $\alpha = 8$ confidence increases monotonically with α.

Dependencies in Time Series

In this section we consider interval sequences that have been extracted from numerical time series. The symbolic description of a time series via intervals of increasing or decreasing trends, convex or concave behaviour, etc., is close to the human's perception of time series. Our data stems from hourly observations of wind strength, wind direction, and air pressure over 7.5 years. The data is suited to test our approach, since one can indeed find rules about short-term weather prediction on the basis of the qualitative/semi-quantitative description of the air pressure curve [Karnetzki, 1999, Sprecher Energie, 1990]. Due to the chaotic nature of the weather, it is impossible for the global weather prediction to exactly forecast when and where strong winds will show up, therefore such rules are used by sailors to make local predictions.

The alternating occurrences of increasing/decreasing and concave/convex segments produces a huge number of rules, for instance "increasing air pressure $\rightarrow_{meets}$ decreasing air pressure" or "concave wind strength $\rightarrow_{meets}$ convex wind strength", and combinations thereof. These rules naturally have high support and confidence. Without any background knowledge these rules would be helpful and are thus ranked high by the J-measure, but they are uninteresting since they are clear from the context (but there is no way of telling the J-measure our background knowledge). Here we have simply filtered out all these rules and consider only cases where the labels in the premise and conclusion address different variables. Alternatively, the background knowledge could be formulated as *a set of known rules* and new rules are printed only if they are not just a combination of known rules.

One has to carefully consider the temporal relationships of the rules when judging about their (subjective) interestingness. For instance, consider a certain premise pattern P and a rule "$P \rightarrow$ increasing wind strength" with a high information content. At first glance the rule might be considered as a good candidate to forecast increasing wind strength. However, if the temporal relationship between the intervals in P and the "increasing wind strength" interval is "before" then it is very likely that P overlaps a "decreasing wind strength" segment right before the "increasing wind strength". In such a case we will also find a strong rule "$P \rightarrow$ decreasing wind strength", which may appear contradictory to the first rule if the temporal relationships are not considered. An example for such a situation is given by $P =$"increasing air-pressure overlaps convex air pressure". The rule "$P \rightarrow_{before}$ decreasing wind strength" has even higher information content than "$P \rightarrow_{overlaps}$ increasing wind strength", but the latter rule is more useful. But again the J-measure is not to blame for this rating, since our preference is based on interpretation issues that are not reflected by the J-measure. We can easily circumvent such situations by requiring during the rule enumeration that the premise and conclusion patterns are *connected*, that is, the rightmost interval of the premise pattern has non-empty intersection with the leftmost interval of the conclusion pattern (cf. section 5.3). Then, only the (subjectively) more interesting rule will be generated. Using these two pruning techniques (different variables in premise and conclusion, connected-

#	temporal pattern	constraints
1	p-dec p-cvx w-inc	supp=7%, conf=59%, J=0.16 bit length(p-cvx)∈[16,37] curv(p-cvx)≤-0.25
2	p-inc p-ccv w-dec	supp=7%, conf=79%, J=0.21 bit length(p-inc)≤80, length(p-ccv)≤41 curv(p-ccv)≥0.06, slope(p-inc)≥0.72
3	d-inc p-ccv w-dec	supp=5%, conf=61%, J=0.13 bit length(d-inc)≤53, length(p-ccv)≥9 curv(p-ccv)≥0.06
4	p-inc p-dec d-inc	supp=19%, conf=85%, J=0.33 bit length(p-inc)≤79, length(p-dec)∈[10,39] slope(p-dec)≤-1.87
5	d-inc d-dec w-inc	supp=4%, conf=62%, J=0.13 bit length(d-dec)∈[7,26], slope(d-dec)≤-0.59 length(d-inc)≤49

Figure 4.5: Exemplary Rules.

ness) the number of rules has been reduced by 90%.

Figure 4.5 shows some exemplary rules. The length of the intervals in the pictorial presentation as well as the length of the gaps are mean values of the interval instances that match the rule pattern. For the labels we use a prefix `w/p/d` for wind-strength/airpressure/winddirection, respectively, and a suffix `inc/dec/ccv/cvx` for increasing/decreasing/convave/convex segments. Intervals that belong to the premise are drawn with thin lines, those that belong to the conclusion with thicker lines. Constraints on the slope refer to the slope of a least-squares line through that segment, constraints on the curvature refer to the mean value of the second derivative.

For the rule #1, for instance, the J-value has almost doubled after specialization. The corresponding air pressure curve has a sharp local minium (the decreasing air pressure segment will meet an increasing segment, the decreasing flank is convex). The minium is restricted not to be too smooth (condition on the curvature). On the average, the wind strength of the upcoming winds will then increase by 3.4 ± 1.7 m/s every hour. Rule #4 tells us that it is very likely that the change in the winddirection will turn from counterclockwise to clockwise after a local maximum in the air pressure curve that has a steep decreasing flank. The few examples show that the obtained rules are easy to interpret and provide useful information about variable dependencies.

We did not create any labels "linear air pressure" (or "constant air pressure") in our interval sequence, but only convex and concave (increasing and decreasing) air pressure. Due to noise in a real-world environment one never has a perfectly constant segment. Nevertheless, it is reasonable to consider the air pressure as constant[2] if it does not change very much. But introducing "linear air pressure" or "constant air pressure" requires to fix a threshold on the slope or curvature, which

[2]Reasonable in the sense that a human perceives such segments as constant and will use such terms when argueing about profiles.

is usually done ad hoc. Instead of "guessing" such thresholds we can hope to get some evidence for them from the specialized rules: if there are similar constraints on the slope variable in different rules, we may use these value to further categorize the increasing segments (e.g., constant, slightly increasing, quickly increasing segments). In rule #2 and #3 the same constraint on the curvature has been identified, it thus would be a candidate threshold for distinguishing "linear air pressure" from convex/concave behaviour. The threshold can then be used to rename the interval labels appropriately and restart the rule discovery process. Since the threshold has been obtained from specialization (but only a small number of rules has been refined) we hope that the information content of many other rules can be increased, because the new set of labels indirectly allows to discriminate with respect to this condition.

Chapter 5

Imprecision and Ambiguity

In this chapter we investigate the problem of imprecision or ambiguity in the abstracted time series (estimated interval bounds and labels). There is not only uncertainty in the segmentation methods, but also uncertainty in the human perception of a time series: Usually, depending on context and focus, a human may address quite different aspects of the same profile. Clearly, this cannot be reflected by considering a single abstraction only. However, it seems that all published approaches use a single representation of the data and therefore ignore the possibility of ambiguity. In this chapter, we discuss the consequences of incorporating ambiguity in the analysis and propose a new algorithm (extension of the algorithm in chapter 3) that handles ambiguity appropriately.

Besides uncertainty in the intervals, there is uncertainty about the duration of intervals and the gaps between them in the temporal patterns themselves (since only qualitative aspects are modelled). While this uncertainty is intended to compensate dilatation and translation effects, under some circumstances it makes certain temporal patterns difficult to interpret (as already pointed out in section 4.3). The *uncertainty in interpretation* makes such patterns less valuable than others, therefore we want to eliminate them in order not to confuse a human analyst. In section 5.3 we consider the question whether leaving out these patterns in the mining step can help to speed up the process.

5.1 Imprecision in Interval Bounds

With most of the methods from chapter 2, the bounds of the segments are to some degree imprecise for various reasons. During function approximation it might be advantegous to slightly shift the position of extrema to decrease the overall error (cf. position of global minimum in figure 2.10 or figure 2.13), with smoothing techniques we also have the effect of dislocation, other methods make use of heuristic decisions, which are not guaranteed to be correct. Thus, with different approaches the extracted interval bounds may slightly deviate from each other and it is unclear which bounds are correct.

A possibilistic or probabilistic modelling of this uncertainty, however, increases the computational cost of support estimation dramatically. (Even the definition of a similarity measure for fuzzy intervals is by no means a trivial task, see e.g. [Cãlin and Gâlea, 2001].) Suppose we have a pattern "*A meets B*". If we are uncertain about the bounds, we have to take the possibility of observing a noisy "*A*

before B" or "*A overlaps B*" into account. Of course, we would not consider each of these possibilities as being equally likely, but would have to define a (possibilistic or probabilistic) "degree of observability", preferrably as a number within $[0, 1]$, to reflect the certainty with which we observe one or the other pattern. The number of patterns we would have to consider may increase quite dramatically if we consider the *equals* relationship: for "*A* equals *B*" imprecision introduces $9 = 3^2$ possible patterns, for a third interval C that *equals* B and A we have $169 = 13^2$ possibilities, and so forth. Compared to the patterns obtained before the consideration of imprecision, a number of additional patterns may be "discovered" just because of the explicit consideration of uncertainty.

Another point is that an instance would have to contribute only *partially* to the support of the respective pattern. For instance, if we say that we have "*A meets B*" with probability 0.8 and only 0.1 for the two other variants "*A before B*" and "*A overlaps B*", then the length of the observation interval should be multiplied with the "degree of observability" before adding it to the support to make sure that a certainly observable instance contributes more to the support than uncertain instances. But this has major consequences for the pruning techniques we have employed in section 3.3. There, we move a pattern that is currently observable into the set of active patterns where it is not considered any longer unless its instance disappears. When observing a pattern only to some degree we are not allowed to proceed in this way in general: For a degree below 1 we still have to seek for other potential instances with a higher degree, because they would contribute more to the support of the pattern. This would make our pruning technique rather inefficient and the whole process much slower.

Therefore, we strongly recommend to use the wavelet multiscale method proposed in section 2.2.3, which is able to compensate the effects of dislocation and flattening of profiles, such that we can be quite sure about correct interval bound estimation.

5.2 Ambiguity in Labels

Let us consider a time series in some neighbourhood around t and assume an increasing trend in this area. This information is probably more reliable than the value at t alone, because a small amount of white noise will not turn an increasing trend into a decreasing trend. There are, however, other effects than white noise, and thus we must be aware that low-frequency disturbances may have caused this increasing trend, but when looking at a coarser scale (zooming out) we have a decreasing trend. While our assumption is that there is always a unique sequence of states a system has run through while producing its output, it is extremly difficult to find the segmentation that corresponds to this sequence, because this means that we have to distinguish all kinds of disturbances to recover the unknown true signal. The difficulty is to distinguish between subtle but important features and uninteresting noise (which is not necessarily Gaussian).

To give another example, consider the time series in figure 2.1 (page 14). There are two major peaks and right before the decreasing flank of the peaks there is a small peak in both cases. Is that a coincidence? Or is it important? Do we want to let a heuristic time series conversion procedure decide about that? Better not, because if the peaks are falsely discarded, even the most powerful pattern detection mechanism will not be able to recover their importance. On the other hand, we do not want to consider every noisy data point separately, since this would increase the computational cost of the pattern discovery process tremendously. There are two different degrees of importance associated with a peak: one is its perceptual

salience (as an a priori indicator for noise or feature), and its importance in the context of a local pattern it may belong to. If such a small peak is always followed by some interesting pattern, then the small peak is of great importance since it can be used for prediction. By imposing a constraint on the lifetime of a segment in scale-space, we can make sure that only perceptually interesting features are considered. The importance with respect to potentially discovered rules, however, can only be revealed by considering all possible interpretations during the process.

If we have a decreasing segment with a noisy bump in it, the segment is divided into three subsegments (cf. section 2.2.3 and figure 2.22): decreasing - increasing - decreasing. Now, within the considered time interval, we have 3 intervals labeled decreasing. Using all 3 decreasing-intervals during pattern discovery is forbidden by the maximality assumption (3.1): two intervals have to be disjoint if they carry the same label, which is not true for any 2 of our 3 intervals if they are taken from different scales. In section 3.3.4 we made use of (3.1) to guarantee that the pattern matching algorithm yields the *earliest* occurrence of a given candidate pattern in the sliding window. Violating (3.1) thus leads to undefined results since the correctness of algorithm 3.10 is no longer guaranteed. Figure 3.24 illustrates this fact with an example. It shows two instances of an example pattern. The instance that comes lexicographically last is observed first, which contradicts Theorem 9 because the maximality assumption of chapter 3 does not hold.

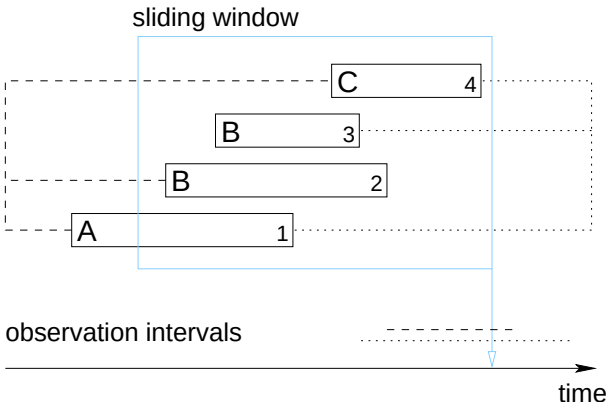


Figure 5.1: Two instances of the pattern "*A overlaps B*, *B overlaps C*". Although $(1, 2, 4)$ (dashed line) is lexicographically before instance $(1, 3, 4)$ (dotted line), the observation interval starts later.

We can circumvent the relaxation of (3.1) by renaming the labels of all intervals such that they reflect the scale from which they have been extracted. A label $s \in \mathcal{S}$ is no longer used alone but only in combination with a scale, like $s - 4$ indicating that the interval with label s holds for scale 4. While "A starts A" is not allowed, "$A - 2$ starts $A - 4$" does not violate (3.1). However, this approach increases the data volume (if an interval labeled A survives over scales 1 to 5, we have to add it 5 times to the interval sequences with label $A - 1$ to $A - 5$) and prevents matching of identical patterns on different scales ($A - 4$ does not match $A - 3$). Therefore, we prefer not to include the scale in the label, that is, to consider only one interval per rectangle in the interval tree of scale. The relative positions of the intervals provide already information about the scales: if one interval contains another with the same label, the larger one must have been extracted from a larger scale (by the tree properties). To follow this approach we have to develop a new subpattern check. The naive approach is to enumerate all occurrences of a temporal pattern and then yield the one that can be observed first, however, this appears to be ineffective due

to the potential combinatorial explosion of possible embeddings.

Let us recall how the visibility of a temporal pattern is determined. A pattern becomes visible if the interval bound next to the rightmost interval bound coincides with the right bound of the sliding window and invisible if the interval bound next to the leftmost interval bound coincides with the left bound of the sliding window (cf. section 3.2). For a given temporal pattern, these points in time will always be determined by the same lower/upper bound of the same intervals, therefore this information may be precalculated and stored together with the pattern. For instance, in the pattern in figure 5.1 it is the left and right bound of the B interval. From the B interval alone we can determine the observation interval of the pattern even without having mapped the whole pattern to the window. In general, two intervals will be necessary for this (e.g. the (left bound of the) first B and the (right bound of the) C interval in figure 3.5 on page 52).

Hence in every temporal pattern two intervals are sufficient for calculating the pattern visibility in the sliding window and determining the observation duration for a given pattern instance is $O(1)$. Appendix A.4 describes the respective algorithm to extract the necessary information (interval number and side) from a temporal pattern. Having this information at hand we can calculate the observation interval (of a potential instance) as soon as we know to which intervals in the sequence these two intervals are mapped. If we know all possible matching interval pairs in the sliding window, we can easily sort these pairs by the expected observation interval. To find the earliest pattern occurrence we search for the first embedding of the pattern using the order that is given by this sorted list of interval pairs. Only little computational overhead is necessary to do this, for every pair of labels (s, t) and temporal relationship r, we maintain a list of matching interval pairs in the sliding window. These lists are kept up to date incrementally whenever the content of the sliding window changes.[1]

Besides that, we use the lists of interval-pairs to influence the backtracking depth of the subpattern check routine. For the remaining intervals of the pattern, we look for the interval-pair with the smallest number of occurrences in the sliding window. The fewer possibilities the lower the chance of intensive backtracking. Rather than trying all suitable intervals from left to right (as it is done in the routine of figure 3.15) our new test routine actively selects the pairs of intervals that will be fixed next in order to (a) yield the earliest instance and (b) limit the maximal number of backtracking steps.

This procedure is closely related to the subpattern test we have proposed on page 58, since again the storage of lists of interval pairs is required. While the sophisticated subpattern check was outperformed by the simple approach (see the evaluation section 3.4), we now use these lists simply to determine the order in which the more simple algorithm (figure 3.10) shall fix the intervals in the pattern.

5.3 Uncertainty in Interpretation of Patterns

With our notion of temporal patterns, interval positions are captured only qualitatively such that repetitions under slightly changed conditions (dilatation or translation) still map to the same temporal patterns. Under some circumstances, however, temporal patterns may be difficult to interpret due to this uncertainty in the inter-

[1] With n intervals in the window the additional space requirement is $O(n^2)$ (same as using a matrix of interval relationships), every removal and insertion of an interval has time complexity of $O(n)$.

val positions. Suppose we have a rule saying when signal A increases steeply then signal B will increase (with temporal relationship *before*). The steep increase in A will be considered as an indicator for an increase in B, but if the window width is considerably large, the gap between both increases may be that long that in fact a non-steep increase of A preceeds the increase in B. If this is the case, it is not very likely that the steep increase in A caused the increase in B. Of course, the gap between the steep increase of A and the increase of B need not to be that large, but we cannot decide about it by looking at the temporal pattern alone.

Hence, such patterns have to be interpreted carefully, since we do not know how much time has passed between the intervals (only that it is less than the sliding window width). The probability of observing both intervals in an *overlaps*- or *meets*-relationship is not affected significantly by the chosen window width: such patterns can already be observed with smaller window width and the number of occurrences increases only slightly as the window width is increased. However, with increasing window width *before*-relationships become significantly more frequent, simply because it is more and more likely that we can observe some more steep increases of A as the width of the sliding window increases – far apart and without any relationship to the increase of B.

This uncertainty in the interpretation of a pattern can be limited by choosing a small sliding window, however, there are also good reasons for selecting a larger sliding window: In most other approaches to knowledge discovery from time series, where the (local) time series similarity is decided with the help of a sliding window, the *complete* content of one window position has to match the *complete* content of another window position to be considered as similar. In our approach, the window is not that restrictive, it mainly provides an upper bound for the temporal extent of patterns, while the content of the sliding window at different positions has to match only *partially* – depending on the pattern we are currently looking for. Therefore, while the complete-match approaches are very sensitive to the chosen window width, the proposed approach is not and therefore it makes sense to insist on larger sliding windows.

We therefore want to ignore patterns where the intervals have such a loose temporal relationship to each other. To capture such patterns, we have the following definition, which is related to the definition of connected patterns (page 74).

Definition 8 *A temporal pattern P is called loosely connected, if the (undirected) graph $G = (V, E)$ is connected, where the set of vertices is given by $V = \{1, .., \dim(P)\}$ and the set of edges is given by $E = \{(i, j) \,|\, R[i, j] \in \mathcal{I}_L\}$ with $\mathcal{I}_L = \mathcal{I} \backslash \{after, before\}$.*

Note that this definition does not exclude after and before relationships in general, for instance in the loosely connected pattern $P :=$"A meets B, B meets C" we have a before relationship between A and C. However, $Q :=$"A overlaps B, B before C, C overlaps D" is not loosely connected, since there is no connection between C and B. Intuitively, a pattern is loosely connected if we can separate parts of the pattern by drawing a vertical line through the patterns without intersecting intervals. Figure 5.2 shows an algorithm to test whether a pattern is loosely-connected or not. We want to restrict the pattern enumeration process to loosely connected patterns. To do this, we have to make sure that during candidate generation all and only loosely connected patterns are generated.

In the current candidate generation algorithm we make use of the fact that any subpattern of a temporal pattern is itself a temporal pattern (used for pruning and candidate generation). This is no longer true for loosely connected patterns. During

```
1  proc loosely_connected(→ P)
2    if dim(P) < 2 then return true fi;
3    allocate array reached of size dim(P), initialize with false;
4    reached[1] ↤ true; n ↤ 1; free ↤ dim(P);
5    while free > 0 do
6          if ¬reached[n] then return false fi;
7          i ↤ n + 1;
8          while i ≤ dim(P) ∧ P.R[n, i] ≠ before do
9                if ¬reached[i]
10                 then reached[i] ↤ true; free ↤ free − 1;
11               fi
12               i ↤ i + 1;
13         od
14         n ↤ i;
15   od
16   return true
17 .
```

Figure 5.2: Testing a normalized temporal pattern P for being loosely connected. The algorithm is linear in the size of P. In lines 6 and 8 we make use of the fact that P is normalized.

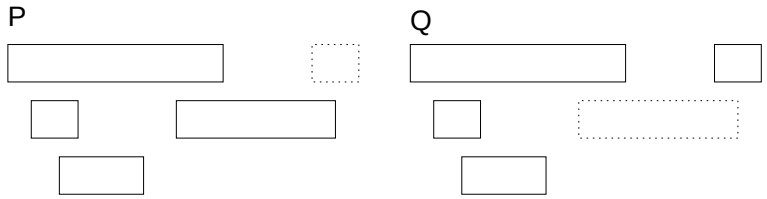


Figure 5.3: The property of being loosely connected persists if the last interval is removed (pattern P) but may get lost if the interval next to the last is removed (pattern Q).

candidate generation, for any $(k+1)$-candidate X we have two k-subpatterns P and Q that share a common $(k-1)$-prefix (those patterns obtained from the candidate by removing one of the last two intervals). For this reason we can expect any new candidate to have two k-subpatterns of it within the same block (cf. page 64), which is used for efficient candidate enumeration. With loosely connected patterns the subpatterns P and Q are not necessarily loosely connected (cf. figure 5.3), and thus it is not sufficient to consider only the patterns in the same block of P to combine them to a new candidate X. Fortunately, we have at least the following observation:

Theorem 11 *Given a (loosely) connected pattern P with $\dim(P) \geq 2$. Then one can find at least two different loosely connected patterns $Q \sqsubset P$ and $R \sqsubset P$ with $\dim(Q) = \dim(R) = \dim(P) - 1$.*

Proof of Theorem 11: We can build the graph $G = (V, E)$ as in the definition of loosely connected patterns. G is connected and we thus can generate a (single) spanning tree for G. The tree has at least 2 vertices with only a single edge. (In a spanning tree with 2 vertices both have only a single edge. As the number of vertices increases, this number is not decreased since cycles are not allowed in a

tree.) When removing any of these 2 vertices, the remaining tree with $k-1$ vertices represents a spanning tree of a connected pattern of size $k-1$. ■

This observation guarantees that we can use a similar candidate construction as before, which is shown in figure 5.4. The pattern Q (obtained by removing the interval next to the last) is not located in the same block as P if it is not loosely connected. Instead of considering only Q as a subpattern of X, we thus have to consider all $k+1$ possible subpatterns Q_i of X, where Q_i is obtained by removing the i^{th} interval of X (and thus $Q_{k+1} = P$, $Q_k = Q$). Note that none of these Q_i patterns besides Q_{k+1} is fully determined by now, the last row/column is still unknown, so far we have no idea what label and interval relationships for the new interval may be chosen. Thus we cannot search for the k-patterns Q_i but for the $(k-1)$-prefixes of Q_i in the set of frequent patterns. This can be done with binary search in a similar way as it has been done for k-patterns before. Any pattern Q' we find serves as a candidate for completing Q_i (and thus X). The $k+1$-pattern X is thereby determined up to a single interval relationship, which is basically the same situation as with the candidate generation we have used before. The only difference is that the missing relationship was between interval k and $k+1$ before, but now it is between interval i and $k+1$ (reflecting that Q_i has been used to find the completions of X rather than $Q = Q_k$).

To do binary search we have to make sure that the generated candidates obey the order of temporal patterns as defined on page 63. While the generated candidates are automatically in this order when using the algorithm of figure 3.15, this is no longer true for the new candidate generation algorithm. It may even happen that a single candidate is enumerated more than once. To restore the order of the candidates we collect all suffixes (last column of the Q' patterns) in an ordered set that does not allow duplicate entries (set of extensions $\mathcal{E}$ in figure 5.4). The order that is used for the suffixes guarantees that the completions of X with any of these suffixes will be in the same order as the suffixes themselves, we therefore build the candidates in the same order as their suffixes are stored in the suffix set.

The suffix vector is a $(\dim(P)+1)$-vector and contains the unknown relationship between interval i and $k+1$. Since sorting of suffixes is not possible with such an *unknown* field, we insert one suffix for each of the 7 possible interval relationships in normalized form – as long as the law of transitivity permits such an interval relationship.

Theorem 12 *The time complexity of algorithm 5.4 is*

$$O(k|F_k||\mathcal{B}|(\log|F_k| + k^2 + \log|\mathcal{B}| + L))$$

where $|\mathcal{B}|$ is the maximum block size (in a block of k-patterns all patterns share a common $(k-1)$-prefix, cf. page 64).

Proof of Theorem 12: Assume that F_k is given and C_{k+1} has to be determined.

In the *build_extensions* routine the outer for-loop is executed k times. Each time, a new subpattern C is created ($O(k^2)$). Then, binary search within the sorted frequent patterns F_k is used to find the first pattern index bp such that C is its prefix ($O(\log|F_k|)$). At most $|\mathcal{B}|$ steps are necessary to find the end ep of the block $[bp, ep]$. The inner loop is then executed $|\mathcal{B}|$ times and consists of a pattern generation step ($O(k^2)$), application of the law of transitivity $O(7k)$ and an insertion into an ordered set $\mathcal{E}$ ($O(\log|\mathcal{E}|)$). Thus, the total runtime is

$$\begin{aligned} & O(k \cdot (k^2 + \log|F_k| + |\mathcal{B}| + |\mathcal{B}| \cdot (k^2 + 7k + \log|\mathcal{E}|))) \\ = \; & O(k^3 + k\log|F_k| + k|\mathcal{B}|(k^2 + \log|\mathcal{E}|)) \end{aligned}$$

```
1  proc candidate_generation(→ F_k, → (O_P)_{P∈F_k}, ← C_{k+1}, ← (E_P)_{P∈C_{k+1}})
2    C_{k+1} ↤ ∅; n ↤ 0;                                   n is actual value of |C_{k+1}|
3    for i ↤ 1 to |F_k| do
4        build_extension_list(F_k[i], F_k, E);
5        enumerate_candidates(F_k[i], E, F_{k+1});
6    od
7  .
9  proc build_extension_list(→ P, → F_k, ← E)
10   E ↤ ∅;                                                E is an ordered set, see text
11   for t ↤ 1 to dim(P) do
12       let C be the (dim(P) − 1)-subpatt. of P with interval #t removed;
13       find index range [bp, ep] where C is prefix of all F_k[j], j ∈ [bp, ep]
14       for i ↤ bp to ep do
15           let X be a (dim(P) + 1)-superpattern of P where the missing
16               last row/col is taken from the last row/col of F_k[i], only the
17               relationship of interval #t and #(dim(P) + 1) is unknown
18           let suf be the last column in X, suf[t] remains undetermined;
20           determine set I of interval relationships that do not violate
21               the law of transitivity if imputed in X.R[t, dim(P) + 1]
23           for each r ∈ I complete suf and insert it in E
24       od
25   od
26 .
28 proc enumerate_candidates(→ P, → E, ← C_{k+1})
29   foreach suf ∈ E do                                    respect order of elements in E
30     extend P to X by adding one row/col given by suf;
31     if loosely_connected(X)
32       then
33           subpattern test: create all dim(P)-subpatterns S_i of X
34               and verify that ∀i : ¬loosely_connected(S) ∨ S ∈ F_k;
35           prune X if verification failed, otherwise:
36               intersect the support sets O_S and prune X if |O_S| < supp_min;
37           if X has not been pruned, insert it in C_{k+1};
38     fi
39   od
40 .
```

Figure 5.4: Candidate generation for loosely connected patterns. For details of the pruning steps, see also figure 3.15.

In the *enumerate_candidates* routine, we iterate over the elements of $\mathcal{E}$, create a new pattern ($O(k^2)$) and test it for being loosely connected ($O(k)$). The following pruning tests are $O(7k^2+k\log|F_k|)$ and $O(kL)$ where L is the length of the sequence (cf. remarks on page 65). Insertion into F_{k+1} is $O(1)$ since the correct ordering is already guaranteed by the order in $\mathcal{E}$. Thus, the total runtime is

$$\begin{aligned} & O(|\mathcal{E}|(k^2+k+7k^2+k\log|F_k|+kL)) \\ = \; & O(|\mathcal{E}|(k^2+k\log|F_k|+kL)) \end{aligned}$$

The size of the set of suffixes $\mathcal{E}$ is bounded by $7|\mathcal{B}|$.

Both subroutines are called for each element in F_k and thus we have

$$\begin{aligned} & O(|F_k|(k^3+k\log|F_k|+k|\mathcal{B}|(k^2+\log|\mathcal{B}|)+|\mathcal{B}|(k^2+k\log|F_k|+kL))) \\ = \; & O(|F_k|(k^3+k\log|F_k|(1+|\mathcal{B}|)+k^2|\mathcal{B}|(k+1)+k|\mathcal{B}|\log|\mathcal{B}|+k|\mathcal{B}|L)) \\ = \; & O(k|F_k|(\log|F_k||\mathcal{B}|+k^2(1+|\mathcal{B}|)+|\mathcal{B}|\log|\mathcal{B}|+|\mathcal{B}|L)) \\ = \; & O(k|F_k||\mathcal{B}|(\log|F_k|+k^2+\log|\mathcal{B}|+L)) \end{aligned}$$

■

Apparently, the new candidate generation algorithm is less efficient than the original one (cf. Theorem 4, page 64), because it has less information to create the candidates (not every subpattern of a loosely connected pattern is loosely connected). Besides the additional effort of maintaining a sorted set of suffixes $\mathcal{E}$ (additional complexity $k|F_k||\mathcal{B}|\log|\mathcal{B}|$) the difference between Theorem 4 and 12 is that $|\mathcal{S}|$ is replaced by $|\mathcal{B}|$: In the original version, the considered block size is bounded by $7|\mathcal{S}|$ whereas in the new version it is bounded by $7|\mathcal{B}|$. Theoretically, the number of possible suffixes is $|\mathcal{S}|\cdot 7^k$: $|\mathcal{S}|$ possibilities for the label and 7 possibilities for each of the k interval relationships. However, as we have already seen in figure 3.12 (page 60) the number of valid temporal patterns is much smaller than the theoretical number of possibilities to fill the relation matrix. Therefore, we expect the runtime for candidate generation to go up by some limited extent, but on the other hand hope that this will be compensated by the reduced number of candidates that have to be considered during support estimation.

Some remarks with respect to rule generation are indispensable at this point. Consider a loosely connected pattern P. Due to its normalized form, the intervals are ordered in time, hence, an "interval on the left" connects to an "interval to the right" by having a non-empty intersection. This means that the removal of the rightmost intervals does not destroy the property of being loosely connected for the remaining pattern. This is important for rule evaluation, where we need both, the support of the premise pattern and the support of the rule pattern. If the premise pattern would not be loosely connected, it is impossible to evaluate the respective rule. The reverse, however, is not true, the conclusion pattern is not necessarily loosely connected. This means that we have to determine the support of the conclusion pattern separately in order to evaluate rules $C \rightarrow R$ as discussed in section 4.1.3.

Restriction to connected patterns. Since we have discussed connected patterns in section 3.3.4 we would like to briefly address the possibility of restricting the search space to connected patterns, which is a subset of loosely connected patterns. What we have discussed for loosely connected patterns holds similarly for connected patterns, that is, with only minor modifications of the algorithm in figure 5.2 and 5.4 the restriction to connected patterns can be implemented.

This restriction is very interesting in the context of support definition: Theorem 6 tells us that instances of connected patterns are always disjoint. The difficulties in *counting pattern occurrences* as we have discussed in section 3.2 and also in section 4.2 is due to the fact that with temporal patterns it is not a priori clear which intervals shall form a pattern instance. But obviously there is no such ambiguity with connected patterns. If the maximality assumption holds it would thus be possible to define the support of connected patterns as the number of pattern occurrences, which would make the sliding window obsolete. While this is interesting to observe, we will not follow this idea, because we think that connected patterns are probably too restrictive to yield interesting results.

5.4 Evaluation

Effect of the restriction to loosely connected patterns. First we want to compare the mining of loosely connected patterns with the results we obtained in section 3.4. We have used the same data set type 3 and 3% minimum support as before (varying window width). The previously obtained results with this setting are shown in figure 3.28 on page 82, the new results are shown in figure 5.5.

The major differences to the experiments shown in section 3.4 are

- The total number of discovered patterns is much smaller when mining loosely connected patterns: as the window width increases (8/12/16) only 50%/14%/5% of the frequent patterns we have discovered before are loosely connected. This strongly supports the claims of section 5.3. Consequently, even for a window width $\Delta t_{win} = 24$ the algorithm needs only about half of the time that was previously necessary for $\Delta t_{win} = 16$.
- The pruning techniques are less efficient, that is, the ratio of frequent patterns among candidate patterns decreases. This is due to the fact that we have fewer loosely connected subpatterns that can be used for pruning. This has major consequences for $k = 3$ where the number of candidates is much larger than before (only about 3% of the candidates become frequent), but is of minor importance for larger values of k. For larger k it is more likely that more than only 2 subpatterns are loosely connected and pruning becomes more efficient.
- Among the three different pruning techniques the third one becomes the most important. It is the most detailed (and time consuming) check and is capable of identifying candidates as non-frequent even if the other checks failed. In figure 3.28 the third pruning technique pruned 10-20% of the patterns and now we have 25-50%.

In our experience, the number of intervals in a multiscale description is 2-3 times the number of intervals in a single scale description. Increasing the number of intervals while keeping the set of labels constant may cause a dramatic increase in the number of frequent patterns (much more relationships contains, starts, finishes, etc.). But only a small percentage of the frequent patterns are loosely connected patterns (percentage decreases drastically with increasing window width), therefore the reduced pruning efficiency in case of loosely connected patterns is compensated by the savings during support estimation.

Effect of ambiguous labels. To see the effect of handling ambiguity in the labels we have examined about 7.5 years of hourly measured wind-strength (prefix

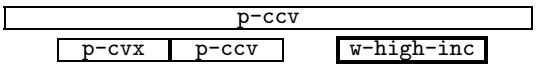
k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	36	100%	36	0	0	0	0.38	0
2	2289	33%	6930	0	0	0	29.16	0.12
3	178	0.7%	25658	674952	144258	1006584	12.81	11.65
4	0	0%	10	4729	537	489	0.51	0.02
$\sum$	2503	7.6%	1.8%	36.5%	7.8%	53.9%	54.65	
1	36	100%	36	0	0	0	0.36	0
2	3318	47.9%	6930	0	0	0	33.45	0.09
3	1410	1.6%	87204	1410731	221745	1989478	54.84	19.59
4	111	72.1%	154	89520	19314	8414	0.89	0.25
5	0	0%	0	2424	261	262	0.47	0.01
$\sum$	4875	5.1%	2.5%	39.2%	6.3%	52%	109.95	
1	36	100%	36	0	0	0	0.36	0
2	3819	55.1%	6930	0	0	0	37.03	0.06
3	5171	2.8%	183256	1883471	250584	2542654	119.93	23.83
4	287	68.2%	421	788082	213285	69531	1.69	2.04
5	1	20%	5	6794	720	741	0.53	0.02
6	0	0%	0	1	0	0	0.46	0
$\sum$	9314	4.8%	3.2%	45%	7.8%	44%	185.95	
1	36	100%	36	0	0	0	0.35	0
2	4203	60.6%	6930	0	0	0	41.21	0.05
3	11269	3.2%	347378	2302443	259971	2951930	227.22	27.41
4	714	52%	1374	2682782	838118	232123	2.87	7.56
5	33	56.9%	58	16664	1961	1656	0.65	0.05
6	0	0%	0	136	5	15	0.49	0
$\sum$	16255	4.5%	3.7%	51.9%	11.4%	33%	307.86	
1	36	100%	36	0	0	0	0.35	0
2	4513	65.1%	6930	0	0	0	47.07	0.04
3	18540	3%	611070	2678592	262083	3194217	405.34	31.01
4	2201	49%	4495	5417018	1860766	477356	4.58	16.8
5	144	69.6%	207	64359	11375	5625	0.94	0.19
6	0	0%	0	1537	114	184	0.52	0.01
$\sum$	25434	4%	4.3%	55.9%	14.6%	25.2%	506.85	

Figure 5.5: Experiments with (from top to bottom) increasing window width (8, 12, 16, 20, 24).

w), air-pressure (p), and wind-direction (d) data. At the beginning, we have used only a few labels for increasing (inc), decreasing (dec), convex (cvx) and concave (ccv) segments, which were refined by additional constraints on gradients or interval lengths later (e.g. p-high-inc means highly increasing air pressure). We used a window width of 48 hours. We did not consider labeled intervals (b, f, s) from very coarse scales that have a length $f - b$ larger than three times the sliding window width. We do this to avoid that almost any frequent pattern P occurs again as "s contains P".

As expected, we obtained much stronger rules than before. To some extent this may be caused simply by the increased number of intervals in the sequence. However, since J-values for some rules have almost doubled, we assume that the representations we used before was not successful in extracting always the 'best' segmentation, which is now recovered by the multiscale method of section 2.2.3.

It turned out that ambiguous features are not only helpful to avoid a "wrong segmentation", but also for the shape description of time series itself. For instance, in the rule



(intervals in the conclusion have a bold frame) it is helpful to know about a more global trend of the air pressure (upper p-ccv interval) to get a better impression of the described time series.

Artificial Example. We demonstrate the effect of considering ambiguity using an artificial example, which has been generated by concatenating noisy squared and unsquared sine waves ($\sin(\frac{2\pi}{l}t)$, $t \in \{0, .., l\}$ with l varying randomly within 200 and 300). For some $\tau \in \{\frac{1}{16}, \frac{7}{16}, \frac{9}{16}, \frac{15}{16}\}$ we randomly added a Gaussian bump ($\exp(-(t - \tau * l)^2/100)/h$ with h varying randomly within 2 and 3). The sine wave is squared if a Gaussian bump appears at $\tau = \frac{1}{16}$. Figure 5.6 shows an excerpt of this time series. The difficulty is to distinguish the important bump ($t = \frac{1}{16}$) from those that have no special meaning ($\tau \in \{\frac{7}{16}, \frac{9}{16}, \frac{15}{16}\}$). It is not possible to generate a single abstraction of the time series that contains only the important bumps, since all bumps have the same characteristics. Their importance can only be revealed by means of the rules that can be discovered by using them.

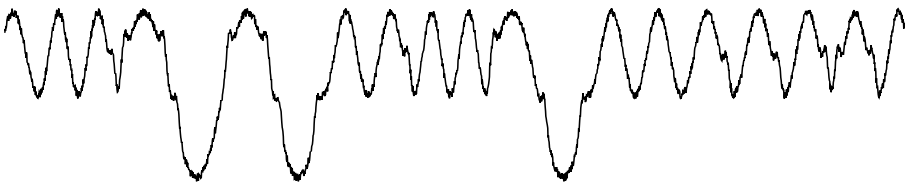


Figure 5.6: The "sine wave with bumps" example.

We ran the discovery process (window width 200, $\text{supp}_{min} = 5\%$) with increasing and decreasing segments only. Then we performed specialization of the best discovered rules to get some evidence for useful thresholds on segment lengths. From the histogram of thresholds on interval lengths we identified two major peaks at segment length ≈ 50 and ≈ 90. Therefore the process was restarted with refined labels denoting the length of the segment (label prefix "short" if ≤ 50, label prefix "long" if ≥ 90). We specialized the discovered rules that contained a "long-dec" label and the best rule with "long-dec" in the conclusion was

> If short-dec | inc has been observed with a gradient between 0.006-0.009 and a length $\geq$ 24 for the inc segment, then short-dec | inc | long-dec will be observed.

The rule has a support of 10% and confidence of nearly 100%, its J-value is 0.34. The premise contains a short decreasing segment (from a Gaussian bump) that meets an increasing segment of the remaining sine wave. A long decreasing segment ($\geq$ 90) is only obtained for an unsquared sine wave, therefore this rule recognizes the relationship between the first bump and the squaredness of the sine curve. The good confidence value is due to the fact that the squared and non-squared sine waves distinguish slightly in their derivative, and the additional quantitative constraint on the gradient[2] focuses on the non-squared sine wave.

This rule can only be obtained by using a multiscale description of the time series. For the long decreasing flank it is important that the midpart of the unsquared sine wave (within $[l/4, 3l/4]$) is always recognized as a single long decreasing segment, regardless whether there are noisy bumps at $\tau \in \{\frac{7}{16}, \frac{9}{16}\}$ or not. On the other hand, at the beginning of the sine wave (within $[0, l/4]$) it is important to distinguish between the occurrence and absence of a bump.

[2]Gradient has been extracted from a linear approximation of the segment.

Chapter 6

Discovery of Meaningful Episodes

Since a pattern has to satisfy the minimum support threshold only to qualify as a rule pattern, it is very likely that many rules are introduced due to noise or incidental co-occurrences of intervals. In this chapter we are concerned about how to reduce the data volume (patterns and rules) that is presented to an expert for verification. We briefly review the problem, known solutions and related extensions of association rule mining, before we finally propose a completely new approach to distinguish interesting from noisy sequences. In particular, we observe that the pattern space itself may be responsible for many patterns that appear incidental at first glance, because of its inadequacy to capture *all* dependencies that may be hidden in the data. This is not a drawback of the chosen pattern space in particular, but is caused by the bias any learning algorithm must have in order to generalize successfully from examples [Mitchell, 1997]. To overcome this limitation, we will consider a new approach to derive *disjunctive combinations* of patterns to approximate the true dependencies more closely.

6.1 Meaningful Episodes

One problem with association rule mining techniques (regardless of whether they are applied to itemsets, event sequences, calendar patterns, or interval sequences) is the size of the rule set, which is often too big to be scanned and evaluated manually. Finally, an expert of the field has to decide whether a rule is incidental, well-known, or indicates something potentially new. Providing too many rules to the expert overburdens him or her quickly – and thus limits the usefulness of rule mining for knowledge discovery.

In the literature, many suggestions can be found how to reduce the number of rules and patterns. The most simple approach is to increase the minimum threshold for pattern support or rule confidence, however, in many cases one obtains only trivial and well-known associations then. A number of approaches make use of additional user information to prune or simplify the set of rules (see for instance [Klemettinen et al., 1994] and the references therein). In this work, we have used the J-measure (discussed in chapter 4) to rank the rules and present only the top rules.

A major problem is that incidental occurrences (above the minimum support threshold) introduce many new patterns and even more rules. Thus, the patterns induced by true dependencies in the data (those we want to discover) are sometimes flooded by incidental patterns. Due to interdependencies of the intervals the derivation of statistical tests is not straightforward. Recently, contingency tables have been used for interval sequences [Cohen, 2001] to decide about the statistical significance of co-occurrences. While a statistical approach is highly desirable, contingency tables lack an underlying statistical model and therefore may behave quite similar to simple pessimistic thresholding techniques.

We have illustrated in section 1 that we assume repetitions of certain patterns in the observed time series. One could therefore argue that the discovery of such repeating patterns (or subsequences) is our main concern, rather than enumerating rules. Indeed, from a single frequent pattern (or sequence) the number of generated rules increases exponentially with the length of the pattern, thus, concentrating on the patterns seems to be much less effort. Moreover, the interestingness of a rule depends on the variables it uses: The best-rated rule is not necessarily most helpful for the expert since it may use unknown variables in the premise or predict well-known variables. By providing the sequences, an expert can decide on his own which intervals he wants to have in the premise and which ones in the conclusion before considering the derived rules.

Although we have fewer patterns than rules, it is still true that for every k-pattern we have $2^k - 1$ subpatterns, thus, the number of subpatterns is still large. Some authors consider only patterns that are maximal, that is, there is no other frequent pattern that contains it as a subpattern (e.g. [Ahonen-Myka, 1999]). While this reduces the number of patterns significantly, we obtain for every incidental occurrence of an interval X in the vicinity of a maximal pattern a new maximal pattern. Such incidental occurrences do not add any information to the maximal frequent patterns, therefore it seems to be more promising to concentrate on the *maximal patterns among the interesting or meaningful patterns* (without having yet defined interestingness or meaningfulness for sequences).

Which sequences are meaningful? It is safe to assume that not every frequent pattern corresponds to a meaningful repetition of some internal states in a system. Following an idea by Cohen and Adams [Cohen and Adams, 2001], a meaningful sequence, called episode in the following, is characterized by the fact that the sequence gets more and more deterministic: At the beginning of an episode, we are not quite sure what we will observe next, but the more elements of the episode we have seen, the more we are certain about what kind of episode we are currently observing (or in which state the system currently is) and it becomes easier to predict the next observation. At the end of an episode we are again uncertain how to continue. In [Cohen and Adams, 2001] entropy is used to measure the goodness-of-fit as the sequence develops, thus the end of an episode is recognized by an increase in the entropy. Cohen and Adams have experimentally verified their approach by removing blanks and punctuation from text (in various languages) and discovering the word bounds quite successfully using this method.

Therefore this appealing idea seems to be helpful in distinguishing incidental patterns from meaningful episodes. Entropy is maximized if all possible continuations of the sequence are equally likely, but the a priori probability of the intervals is not necessarily uniform. Cohen and Adams perform some normalization to cope with this. We simply apply the goodness-of-fit term j of the J-measure for this purpose. By considering episodes rather than sequences we are more robust against incidental occurrences of intervals. Since we do not want to consider such incidental patterns during generalizations of rules, we use the set of episodes rather than the

set of frequent sequences for further processing.

6.2 Evaluating Episodes

We are still lacking an episode evaluation measure. From their definition we know that the goodness-of-fit increases as the episode becomes longer. We want to rate episodes by their suitability to create strong rules out of them, and thus goodness-of-fit alone is not suited to measure the usefulness of an episode, the average applicability of a rule is also important (as we have discussed in section 4).

We want to use the J-values for rules that can be obtained from an episode to rank the episode itself. How many rules can be derived from a k-episode R? We will use the alternative notation $P \rightarrow_R C$ for a rule $P \rightarrow R$ with conclusion pattern C under rule pattern R for notational convenience. Let us divide R into a premise P_i and conclusion part C_i, such that $|P_i| = i$ and $|C_i| = |R| - i$. There are $|R| - 1$ possibilities for this subdivision. Any pair of subsequences (besides the empty sequence) $P' \sqsubseteq P$ and $C' \sqsubseteq C$ will do for a potential rule $P' \rightarrow C'$. We have $|P|$ intervals in the premise and $|C|$ intervals in the conclusion, and thus for each subdivision $(2^{|P|} - 1) \cdot (2^{|C|} - 1)$ rules. Which rules shall we use to rank the episode then?

If we simply use the maximum of all J-values of all rules

$$J_R = \max_{i \in \{1,..,|R|-1\}} \max_{C' \sqsubseteq C_i} \max_{P' \sqsubseteq P_i} J_{P' \rightarrow_R C'} \tag{6.1}$$

it is impossible to distinguish between different developments of the J-value with an increasing length of the sequence. Two sequences R and R' get the same ranking even if the maximum J-value is obtained for different points of subdivision i. But smaller values of i are preferable (given the same J-value), because then a shorter prefix is necessary to reliably predict the continuation of the sequence. Therefore, rather than using a single number we use a tuple of $|R| - 1$ J-values to rank an episode, such as

$$J_R^+ = \left(\max_{C' \sqsubseteq C_i} \max_{P' \sqsubseteq P_i} J_{P' \rightarrow_R C'} \right)_{i \in \{1,..,|R|-1\}} \tag{6.2}$$

Now, $J_R^+[i]$ denotes the J-value of the best subrule of $P_i \rightarrow R$. (Note that $J_R^+[i]$ is not necessarily increasing with i as the goodness-of-fit does.)

We are not quite satisfied with this, because a strong relationship between a short i-prefix of R and the 1-suffix of R would yield consistently high J-values for all $J_R[i']$ with $i' > i$, because for $i' > i$ there is always a subrule that contains the i-prefix in the premise and the 1-suffix in the conclusion. Therefore, our final choice for the episode evaluation is an $(|R| - 1)$-tuple such that $J_R[i]$ yields the minimal J-value that can be obtained from a subpattern of P_i for *any* conclusion $C' \sqsubseteq C_i$:

$$J_R^- = \left(\min_{C' \sqsubseteq C_i} \max_{P' \sqsubseteq P_i} J_{P' \rightarrow_R C'} \right)_{i \in \{1,..,|R|-1\}} \tag{6.3}$$

A ranking of episodes can be obtained by sorting the $J_R^-[i]$ values in decreasing order and comparing the tuples (of varying size for varying length of the episode) lexicographically. Note that $[J_R^-[i], J_R^+[i]]$ provides an interval of J-values in which any rule (with any conclusion $C' \sqsubseteq C_i$) and optimized premise $R' \sqsubseteq R_i$ will fall.

Before we consider the calculation of $[J_R^-, J_R^+]$ let us investigate the J-values we may obtain from rules more closely.

Theorem 13 *Given a premise pattern P and rule pattern R.*

1. *Among the rules $P \rightarrow R'$ with $P \sqsubseteq R' \sqsubseteq R$ the highest J-value is obtained by a rule pattern of size $|P| + 1$.*
2. *Among the rules $P \rightarrow R'$ with $P \sqsubseteq R' \sqsubseteq R$ the lowest J-value is obtained by $R' = R$.*

Proof of Theorem 13: Figure 6.1 shows a graph of $J(P \rightarrow R)$ versus the rule pattern probability $Pr(R \sqsubseteq W) = Pr(x)$ and premise pattern probability $Pr(P \sqsubseteq W) = Pr(y)$. In the following, we will show that J is monotonically increasing with $Pr(x)$ for a fixed premise $Pr(y)$.

For part (1) of the Theorem we can thus conclude that the highest J-value is obtained if the probability of observing the rule pattern $Pr(x)$ is maximized. From Theorem 3 we know that the support (and thus the probability of occurrence) of a pattern decreases as its complexity or dimension increases. Therefore, the highest probability $Pr(x)$ is obtained from the simplest rule pattern possible, and thus $|R| = |P| + 1$.

Part (2) of the Theorem is shown using the same arguments. If a (complex) rule pattern R is given, the dimension of a subpattern $R' \sqsubseteq R$ is smaller or equal to that of R, therefore its probability of occurrence is greater or equal than that of R. Due to the monotonicity the lowest J-value is obtained from the rule pattern with lowest probability of occurrence.

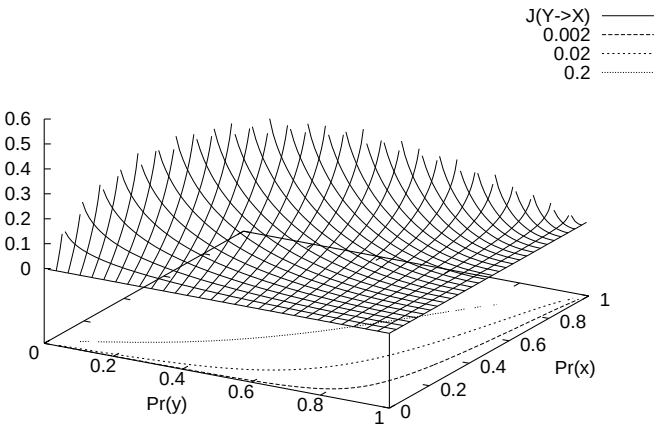


Figure 6.1: Information content of a rule.

The monotonicity remains to be shown. Given a rule $Y \rightarrow X$, using the abbreviation $Pr(x|y)$ for $Pr(X = x|Y = y)$ the J-measure is given by

$$\begin{aligned} J(X|Y=y) &= Pr(y) \cdot \sum_{z \in \{x, \bar{x}\}} Pr(z|y) \log \left(\frac{Pr(z|y)}{Pr(z)} \right) \\ &= Pr(y) \cdot \left(Pr(x|y) \log \left(\frac{Pr(x|y)}{Pr(x)} \right) + \right. \\ &\qquad \left. (1 - Pr(x|y)) \log \left(\frac{1 - Pr(x|y)}{1 - Pr(x)} \right) \right) \end{aligned}$$

since $Pr(\bar{x}) = 1 - Pr(x)$ and $Pr(\bar{x}|y) = 1 - Pr(x|y)$. With $Pr(x|y) \cdot Pr(y) = Pr(x, y)$ we continue

$$\begin{aligned} J(X|Y=y) &= Pr(x,y) \log\left(\frac{Pr(x,y)}{Pr(x)Pr(y)}\right) + \\ &\quad (Pr(y) - Pr(x,y)) \log\left(\frac{1 - Pr(x|y)}{1 - Pr(x)}\right) \end{aligned}$$

Since the rule pattern comprises the premise pattern, we have $Pr(x, y) = Pr(x)$ and obtain

$$\begin{aligned} J(X|Y=y) &= Pr(x) \log\left(\frac{1}{Pr(y)}\right) + \\ &\quad (Pr(y) - Pr(x)) \log\left(\frac{Pr(y)}{Pr(y)} \cdot \frac{1 - Pr(x|y)}{1 - Pr(x)}\right) \\ &= Pr(x) \log\left(\frac{1}{Pr(y)}\right) + \\ &\quad (Pr(y) - Pr(x)) \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) \end{aligned} \tag{6.4}$$

Let us now assume that $Pr(y)$ is fixed and calculate the partial derivative of J with respect to $Pr(x)$:

$$\begin{aligned} \frac{\partial J(X|Y=y)}{\partial Pr(x)} &= \log\left(\frac{1}{Pr(y)}\right) - 1 \cdot \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) + \\ &\quad (Pr(y) - Pr(x)) \left(-\frac{1}{Pr(y) - Pr(x)} - \frac{-Pr(y)}{Pr(y)(1 - Pr(x))}\right) \\ &= -\log\left(\frac{Pr(y) - Pr(x)}{1 - Pr(x)}\right) + \left(-1 + \frac{Pr(y) - Pr(x)}{1 - Pr(x)}\right) \end{aligned} \tag{6.5}$$

Now, consider the function $f(z) = z - \log(z)$ for $z \in (0, 1)$. For $z \to 0$ we have $\lim_{z \to 0} z - \log(z) = \lim_{z \to 0} \log(1/z) = \infty$ and for $z = 1$ we obtain $1 - \log(1) = 1 - 0 = 1$. The function f is monotonically decreasing for z.[1] Together with the values of f near the bounds we obtain $z - \log(z) > 1$ or $-\log(z) + (-1 + z) > 0$. Now, consider $z := \frac{Pr(y) - Pr(x)}{1 - Pr(x)}$, for which we have $z \in [0, 1]$ thanks to $Pr(y) > Pr(x)$. Substituting z in $-\log(z) + (-1 + z) > 0$ as described yields the desired monotonicity of (6.5). ■

Fortunately the calculation of the $[J_R^-, J_R^+]$ intervals can be done efficiently:

Theorem 14 *Given a set of all frequent patterns $\mathcal{F}$, the calculation of the $[J_R^-, J_R^+]$ intervals for all episodes $R \in \mathcal{F}$ is*

$$O(|\mathcal{F}| \cdot K \cdot (\log |\mathcal{F}| + K))$$

where K denotes the maximum size of a pattern in $\mathcal{F}$.

Proof of Theorem 14: In a first step, we separate episodes from patterns using the algorithm shown in figure 6.2. For every k-pattern $P \in \mathcal{F}$ we will store $(k-1)$-vectors of J-values $P_{minJ}[\cdot]$ and $P_{maxJ}[\cdot]$ (which remain identical in this first part of the

[1] f decreases within $(0, 1)$ due to $\lim_{z \to 0} f(z) > f(1)$. To be increasing somewhere in $(0, 1)$ there must be a local minimum within $(0, 1)$, but the only zero crossing of $f'(z) = 1 - 1/z$ is at $z = 1$. Thus f must be monotonically decreasing.

algorithm). We assume that $\mathcal{F}$ is sorted such that the prefix property is preserved (if P is a prefix of Q then P comes before Q, see page 63), sorting is $O(|\mathcal{F}| \log |\mathcal{F}|)$. Then we identify episodes from patterns as described in the previous section by running once through the patterns in this order (`do-while` loop in lines 5-15 in which elements R are fetched from $\mathcal{F}$ sequentially in lines 11 and 13). At any time we keep all i-prefixes $Q[i]$ of a k-pattern R and initialize $R_{minJ}[i] = R_{maxJ}[i] = J(R|Q[i])$. As long as the goodness-of-fit increases (line 7, the frequent pattern is considered as an episode and inserted in $\mathcal{E}$ (line 9). For each k-episode, we fill the $P_{minJ}[\cdot]$ and $P_{maxJ}[\cdot]$ arrays, which is $O(k)$. Thus, the first phase is $O(|\mathcal{F}| \log |\mathcal{F}| + K|\mathcal{F}|)$.

```
 1 let F be a sorted list of frequent patterns of size 1 ≤ k ≤ K      O(|F| log |F|)
 2 for P ∈ F let P_maxJ[·] and P_minJ[·] be a (|P| − 1)-tuple
 3 let Q be an empty vector of patterns
 4 fetch first pattern R ∈ F
 5 do                                                                 O(|F|)
 6     k ↤ |R|; Q[k] ↤ R;
 7     if k ≤ 2 ∨ j(Q[k − 1]|Q[k − 2]) < j(R|Q[k − 1])
 8       then
 9             append R to E
10             for i ↤ 1 to k − 1 do R_maxJ[i] ↤ R_minJ[i] ↤ J(R|Q[i]) od
11             fetch next pattern R ∈ F
12       else
13             fetch next pattern R ∈ F until |R| ≤ k
14     fi
15 until all patterns R ∈ F are processed
16 now E is a sorted list of frequent episodes
```

Figure 6.2: Determining episodes among frequent patterns. For every episode R, $R_{minJ}[i]$ (and $R_{maxJ}[i]$) contains the J-value of the rule that is obtained by subdividing the rule pattern R into premise and conclusion at position i.

So far, $R_{minJ}[i] = R_{maxJ}[i]$ denotes the J-value of a rule obtained by subdividing the rule pattern R at position i into premise and conclusion pattern. In the second phase, we will calculate $R_{maxJ}[\cdot]$ and $R_{minJ}[\cdot]$ according to (6.2) and (6.3).

We show by induction that at the end of the inner loop in figure 6.3, lines 3-15, the $R_{maxJ}[\cdot]$ and $R_{minJ}[\cdot]$ arrays are correct for all patterns R with $\dim(R) < k$ (induction hypothesis). Regarding the initialization ($k = 2$), for any 2-pattern R the values $R_{minJ}[1]$ and $R_{maxJ}[1]$ already correspond to $J_R^-[1]$ and $J_R^+[1]$, because only a single rule can be derived from a 2-episode (and its J-value has been determined in algorithm 6.2).

Now consider $k > 2$ and a rule pattern $R \in \mathcal{E}$. For fixed i let us denote the premise and conclusion pattern by P_i and C_i. Then $P_{minJ}[i]$ is given by (6.3):

$$J_R^-[i] = \min_{C' \sqsubseteq C_i} \max_{P' \sqsubseteq P_i} J_{P' \to_R C'}$$

Whatever P' yields the maximum J-value, C' has to be identical to C_i (which is the maximal $C' \sqsubseteq C$) due to Theorem 13, part 2. Thus, we have

$$J_R^-[i] = \max_{P' \sqsubseteq P_i} J_{P' \to_R C_i}$$

By R_l we denote the subpattern of R where the l^{th} interval has been removed, $l < i$. We can rewrite J_R^- recursively

$$J_R^-[i] = \max\{ \quad \max\{J_{R_l}^-[i-1] \,|\, 1 \le l < i\}, \quad J_{P_i \to_R C_i}\}$$

The J-value of $P_i \rightarrow_R C_i$ is stored in $R_{minJ}[i]$ (that was done in the previous phase by algorithm 6.2). The J-values $J^-_{R_l}[i-1]$, $l < i$, are already determined by the induction hypothesis, because $\dim(R_l) < \dim(R)$. Thus, the $R_{minJ}[\cdot]$ array is correctly determined in line 12.

Secondly, $R_{maxJ}[i]$ is given by

$$J^+_R[i] = \max_{C' \sqsubseteq C_i} \max_{P' \sqsubseteq P_i} J_{P' \rightarrow_R C'}$$

which can also be rewritten recursively as

$$J^+_R[i] = \max\{\quad \max\{J^+_{R_l}[i-1] \mid 1 \le l < i\}, \max\{J^+_{R_l}[i] \mid i \le l \le k\}, \quad J_{P_i \rightarrow_R C_i}\}$$

In contrast to the case of J^-_R, where the conclusion pattern had maximum size, here we also consider rules with a reduced conclusion pattern $C' \sqsubseteq C_i$, therefore l covers the complete range from 1 to k. Depending on whether we have removed an interval from the premise or not, we have to decrement the index of the J-array or not to access the correct rule in the $J^+_{R_l}$ array. This calculation is done in line 11. At the end of the inner loop, all arrays have correct values, therefore the induction hypothesis is shown for k.

Here is a scheme that illustrates the cases for an example 4-episode ($k = 4$), the 3 possible subdivision points i and removed intervals l. For $J^+_R[i]$ the complete column is considered, for $J^-_R[i]$ only those entries with a full conclusion pattern (e.g. the top 2 entries in the column $i = 2$).

	$i=1$	$i=2$	$i=3$
$l=1$	not a rule	$B \rightarrow C\,D$	$B\,C \rightarrow D$
$l=2$	$A \rightarrow \quad C\,D$	$A \quad \rightarrow C\,D$	$A \quad C \rightarrow D$
$l=3$	$A \rightarrow B \quad D$	$A\,B \rightarrow \quad D$	$A\,B \quad \rightarrow D$
$l=4$	$A \rightarrow B\,C$	$A\,B \rightarrow C$	not a rule

Regarding the complexity of the second phase, the inner loop is entered $|\mathcal{E}|$ times. For each of the $(k-1)$-subpatterns we have to look it up in the set $\mathcal{E}$ and calculate the new array values, thus we have a total runtime of $O(|\mathcal{E}| \cdot K(\log|\mathcal{E}| + K))$.

Since $|\mathcal{F}| \ge |\mathcal{E}|$ both algorithms together have the complexity $O(|\mathcal{F}| \cdot K(\log|\mathcal{F}| + K))$.

■

6.3 Generalization of Core Episodes

Core Episodes

The J-measure will serve us in ranking the episodes by the average information content of the rules that we may derive from them. But we are not only interested in a sequential ranking of all episodes, but also in reducing the number of episodes that we have to consider. The idea of maximal episodes is that any subepisode can be generated from a superepisode, therefore it makes no sense to consider all subepisodes individually. We have already indicated in section 6.1, that maximal episodes represent large sets of subepisodes, but do not represent the most interesting episodes since they usually contain noise. Can our episode evaluation measure help us in identifying the incidental co-occurrences? Given an episode, from any superepisode we can generate all those rules that can be generated from the episode

```
1  let E be a sorted list of frequent episodes of size 1 ≤ k ≤ K
2  for k ↤ 3 to K do
3      for R ∈ E ∧ |R| = k do                    both for-loops together: O(E)
4          for l ↤ 1 to k do
5              let Q ⊑ R where the l^th interval has been removed
6              if Q ∉ E then skip this for-loop, go to next l fi
7              for i ↤ 1 to k − 1 do
8                  if (¬(l = 0 ∧ i = 0) ∧ ¬(i = k − 1 ∧ l = k))
9                    then
10                       if l < i then ii ↤ i − 1 else ii ↤ i fi
11                       R_maxJ[i] ↤ max(R_maxJ[i], Q_maxJ[ii])
12                       if l < i then R_minJ[i] ↤ max(R_minJ[i], Q_minJ[ii]) fi
13                 fi
14             od
15         od
16     od
17 od
```

Figure 6.3: Evaluating the $J_R^-[i]$ and $J_R^+[i]$ vectors for episodes.

itself. Thus, when considering the maximum J-value that can be obtained according to (6.1) or (6.2), a superepisode cannot have smaller J-values than any of its subepisodes – which is basically the same situation as before where we had no episode measure: Noisy patterns are "preferred" due to the fact that they have more subpatterns. However, this is no longer true when using definition (6.3).

We consider episodes as distinguished if the full episode is needed to obtain the best J-value; that is, there is a subdivision point i and no subrule of $P_i \rightarrow_R C_i$ yields a better J-value. If $P_i \rightarrow_R C_i$ contains incidental intervals we can improve the value of the rule by removing them. But if the best J-value is obtained by using all intervals, it is very likely that all of these intervals are meaningful in this context. Therefore we call such an episode a *core episode*. It provides the core for many maximal superepisodes but the J-value cannot be improved by them. Therefore we assume that the core episodes are close to those patterns that are caused by the repetitive cycling through internal states of a system. The core property can be determined by the algorithm in figure 6.3 if we additionally store a pointer to the pattern that provides $J_{minJ}[i]$. If for some pattern P and some i this pointer leads us to P itself, we have identified a core episode.

With maximal episodes no maximal superepisodes exist (by definition), but core episodes may have superepisodes that are core episodes. This is due to the fact that a long episode may obtain its core property from a subdivision point that is larger than any subdivision point of the subrule. An episode ABC (interval relationships omitted) may be a core episode from $i = 2$, that is, the rule $AB \rightarrow C$ has a better J-value than any of its subpatterns. However, if ABC is a prefix of an even longer episode $ABCDE$, the superepisode may also be a core episode for $i = 3$. Among the set of core episodes we therefore consider only the maximal core episodes (that is, core episodes that have no core superepisode).

Generalization

By now we have identified a set of maximal core episodes, which is much smaller than the set of episodes or even maximal episodes. By definition, every interval in a core episode seems to be meaningful, since it cannot be removed without decreasing the J-value. So, is it sufficient to present only the maximal core episodes to the domain expert? Our answer is yes, given that the pattern space is powerful enough to express all relationships in the data. But we think that this property cannot be guaranteed – except in rare cases – and taking the possibility of an inadequate pattern space into account the answer must be no. We call a pattern space inadequate when we want to express the fact that there may be relationships in the data that are not representable by hypotheses from the pattern space. One can think of many examples where a relationship is decomposed into many different fragments by the pattern space (e.g. "two out of four intervals take place simultaneously", see also section 4.1.4). In this case it is very likely that none of the fragments itself yields a strong rule and thus none of these fragments becomes a maximal core episodes. Nevertheless, such episodes carry valuable information despite their low J-value – if they are considered in the context of similar episodes.

Our approach to tackle this problem is the generalization of episodes to new (disjunctive) longer episodes. In a naive approach to generalization we could extend the association rule mining step to handle disjunctions, that is, enumerate all generalizations that fulfil the conditions on minimum support and confidence. But a temporal pattern of dimension k has k different labels and $k \cdot (k-1)/2$ interval relationships (the remaining interval relationships can be determined uniquely). We would have to generalize potentially any of the $k + k \cdot (k-1)/2$ values for each pattern. Such a "bottom-up" approach to generalization increases the computational burden tremendously and increases the number of rules even further. Our assumption is, that the pattern space has been designed carefully, taking the kind of expected patterns into account. So what we obtain so far are by no means a random fragmentation of the true pattern, but it is very likely that certain aspects of the true relationship can be captured by an episode in the pattern space. From a computational perspective it is much more promising to start generalization from such a near-miss episode than generalizing everything. Our hypothesis is that (subepisodes of) maximal core episodes provide such near-miss patterns.

Before we discuss what we mean by generalization of episodes in greater detail, let us briefly review the relationship to another extension of association rule mining that has been proposed in the literature: For the case of market basket data, product hierarchies or taxonomies have been suggested to generalize items like "orange juice", "apple juice", "cherry juice" simply to "juice" in order to obtain stronger rules. It may be appropriate to distinguish the different kinds of juices for some associations, while it may be better to *generalize* juices in others. These problems have been solved by introducing a priori knowledge about the product taxonomy [Srikant and Agrawal, 1995, Han and Fu, 1995]. However, we cannot be sure that our taxonomy contains all useful generalizations: Do we want to consider "tomato juice" in our *is-a* hierarchy as a "juice" or introduce another level to distinguish between vegetables and fruit juices?

What is the relation of this to the problem of large rule sets discussed in this chapter? The product hierarchies seem to address a completely different phenomenon at first glance. On second thought, the stronger *generalized juice* rules make a number of *specialized orange/apple/cherry juice* rules obsolete. In absence of the juice generalization we have a number of only moderately strong rules and miss the strong "true relationship" in the data due to an inadequacy or incompleteness of

the pattern space. It would be possible to learn the generalized terms automatically, if our pattern space would allow not only simple rules

$$A\,D\,E \to F\,G, \qquad C\,D\,E \to F\,G, \qquad B\,D\,E \to F\,H, \qquad \ldots \tag{6.6}$$

but also disjunctive combinations like

$$(A \vee B \vee C)\,D\,E \to F\,(G \vee H) \tag{6.7}$$

If a taxonomy is given, it may happen that $(A \vee B \vee C)$ matches a generalized term in our taxonomy (but this is not guaranteed, of course). If the *true association* in the data is given by (6.7) or negative patterns "A but no B", and our pattern space does not contain such patterns we will observe *fragments* of the true pattern only – as in (6.6).

As already mentioned, one may want to conclude that the choice of the pattern space should be thought over, however, the pattern space has usually been chosen after carefully balancing its modelling capabilities and the computational cost of searching it efficiently. We do not want to put the pattern space in question, because the consideration of all possible disjunctive combinations would increase the cost of rule mining dramatically. We consider the question of whether it is possible to identify the true patterns by generalization of the fragmented patterns. Specialization and generalization are considered to be two of the most basic techniques used by the brain to generate new rules. While the association rule mining process can be seen as a *specialization step* (every possible rule is enumerated), a second phase of *generalizing* the specialized patterns seems to be worthwhile.

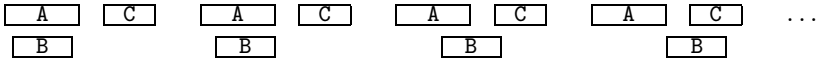
If none of the rules in (6.6) is a core episode, some subepisode probably will be, e.g. $D\,E \to F$, $A\,D\,E \to F$ or $D\,C \to F\,G$. We will use such a core (sub)episode as the starting point for generalization. Again, the J-measure will play an important role in judging the usefulness (or acceptance) of a possible generalization (e.g. to test wether $D\,C \to F\,G$ should be generalized to $(A \vee B)\,D\,C \to F\,G$). It will only be accepted if the J-value of the episode can be improved. Thus, a generalization itself becomes a (generalized) maximal core episode. As a side effect, the existence of a new (longer) core episode prunes other core episodes which are now subepisodes of the generalization. Thereby generalization helps to reduce the number of maximal core episodes as well as improving the rating of the remaining core episodes.

We always select the best-so-far maximal core episode to be generalized next. Once the episode has been generalized, it will not be considered again for generalization. Let R be such an episode. Generalization of R will be done incrementally, starting from superepisodes (not only core superepisodes) of length $|R|+1$, then $|R|+2$, etc. Thus we start with collecting in $\mathcal{S}$ all episodes of length $|R|+1$ that contain R as a subepisode. Since we want to improve the J-rating of episode R, we start from the largest possible subdivision point i down to 1. For each value of i we try to find a subset $\mathcal{G} \subseteq \mathcal{S}$ such that the disjunctive combination of episodes in $\mathcal{G}$ maximizes the J-value. If the maximal J-value is higher than that of the core episodes $J_R[i]$, generalization was successful and $\mathcal{G}$ is considered as a new maximal core episode.

Having found some i and $\mathcal{G}$ such that the J-value is improved over the core episode, we mark the episodes in $\mathcal{G}$ as being part of a generalized core pattern and mark all subpatterns of them, too. This is to exclude maximal core patterns that are subepisodes of the generalized core pattern from further generalization.

Alternatively, we can use generalization to find a subset of patterns in a set of patterns with identical labels but different temporal relationships. In the example (6.6)–(6.7) the size of the patterns varied, but even with a constant set of labels we may observe pattern fragmentation. For instance, if the true relationship is "if

B starts some time after A has started, then will observe C after A", we will find patterns



As a disjunctive combination (generalization of core episode "*A before C*") they can represent the original pattern, but every single one will probably have low J-values. The problem is almost identical to the problem before, we have a set of episodes (with identical labels in this case) and want to select an optimal subset that maximizes the J-value of the disjunctive combination of episodes in the subset.

6.4 Determining the Generalization Efficiently

Given a set of sequences and a subdivision point i, we create a set of rules $\mathcal{S}$. The task is to efficiently determine a subset $\mathcal{G}$ such that the disjunctive combination of the rules in $\mathcal{G}$ maximizes the J-value. We can expect the number of superepisodes of a core episode to be only moderate, but due to the exponential growth of the number of subsets ($2^n - 1$ rule subsets for a set of n rules) a brute force enumeration of all possible subsets is infeasible.

Finding the best subset $\mathcal{G} \subset \mathcal{S}$ requires an estimation of the premise and rule pattern support of the disjunctive combination of episodes in $\mathcal{G}$. We are not allowed to simply add the support values of the single episodes since some sliding window positions may contribute to multiple episodes in $\mathcal{G}$. But another database pass is not required, since during frequent pattern enumeration we have maintained a condensed representation of the pattern occurrences, namely the list of intervals in which the patterns are visible (cf. chapter 3). This representation can be used to unite and intersect support sets efficiently and thus to calculate the J-values of various sets $\mathcal{G}$ more efficiently.

```
 1 proc find_best_subset(→ S[], → i, → P, → R, ↔ bestJ)
 2   for n ↤ i to |S| do
 3       R' = R ∪ S[i].R;                  add support of rule pattern of spec. #i
 4       P' = P ∪ S[i].P;                add support of premise pattern of spec. #i
 6       if i ≠ |S|
 7         then find_best_subset(S, i + 1, P', R', bestJ);
 8       fi
10       j ↤ J(P' → R')
11       if j ≥ bestJ then bestJ = J; valid generalization found; fi
12   od
13 .
```

Figure 6.4: Brute force enumeration of all possible subsets of $\mathcal{S}$.

Figure 6.4 shows a brute force algorithm to find a subset that improves the J-value over the J-value of the core episode by simply enumerating and testing all possibilities. As an example, for a set $\mathcal{S} = \{R_1, R_2, R_3\}$ the subsets $\mathcal{G}_i$ will be

generated in the following order:

$\mathcal{G}_1$	R_1			0.12
$\mathcal{G}_2$		R_1, R_2		0.11
$\mathcal{G}_3$			R_1, R_2, R_3	0.18
$\mathcal{G}_4$		R_1, R_3		0.16
$\mathcal{G}_5$	R_2			0.12
$\mathcal{G}_6$		R_2, R_3		0.11
$\mathcal{G}_7$	R_3			

Making use of an (arbitrary) order of the elements in $\mathcal{S}$ the algorithm makes sure that every possible generalization is enumerated only once. To turn this algorithm into a feasible approach we are in need of a pruning technique that prevents us from checking all subsets individually and thus from the combinatorial explosion in the number of subsets. Simple pruning approaches like "if a rule did not improve the J-value at iteration depth #i it will not improve the J-value at any later depths" will not work, as the following example will show. Suppose the premise and rule pattern support of the rules R_1, R_2, and R_3 is given as shown in figure 6.5. The last column in the table above contains the J-value obtained by the disjunctive combination of these rules. We can see that the J-value of $R_1 \vee R_2$ decreases from 0.12 to 0.11 and the pruning method sketched above would therefore stop searching in this part of the search space and consequently miss the combination $R_1 \vee R_2 \vee R_3$ with the maximal J-value of 0.18.

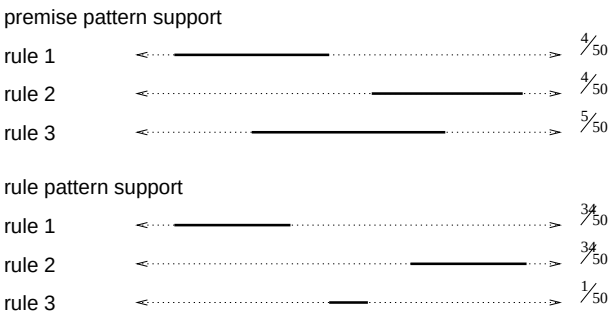


Figure 6.5:

The reason for this failure lies in the fact that the frequencies used for the determination of the J-value are obtained by merging support sets: If the consideration of a specific rule was not helpful at some iteration level i this may be due to the fact that not only the support of the disjunctive rule pattern is increased but also the support of the disjunctive premise pattern. If the ratio of the increase in the rule support and the increase in the premise support falls below a threshold the J-value of the disjunctive rule will not increase. However, after having merged even more rules it may happen that the support of the premise pattern is almost completely contained in the premise support of the disjunctive rule such that the merging yields a different ratio and now the J-value of the disjunctive rule does increase.

To implement a correct pruning technique we have to estimate an upper bound of the J-value of all the rules the algorithm may generate at any recursion depth. To get such an estimate we consider the whole set of possible support values we may obtain in future stages. From this range of support values we will calculate the maximally achievable J-value and if this one remains smaller than the J-value of the core pattern, we can prune this part of the search tree. To get such an estimate, we consider the range of premise and rule support that can be obtained at any node in the search tree as follows: Whenever we consider the i^{th} rule $\mathcal{S}[i]$, in deeper

recursion levels only rules $\mathcal{S}[j]$ with $j > i$ will be considered by algorithm 6.4. The following scheme shows which rules will be considered when further descending the search tree:

				still under consideration
$\mathcal{G}_1$	R_1			R_2, R_3
$\mathcal{G}_2$		$R_1 \cup R_2$		R_3
$\mathcal{G}_3$			$R_1 \cup R_2 \cup R_3$	–
$\mathcal{G}_4$		$R_1 \cup R_3$		–
$\mathcal{G}_5$	R_2			R_3
$\mathcal{G}_6$		$R_2 \cup R_3$		–
$\mathcal{G}_7$	R_3			–

While the current disjunctive combination gives us a lower bound for the premise and rule support in this branch, the union with the premise and rule support of the rules in the last column gives us an upper bound. Together, both ranges give us a rectangle in the premise-/rule-support plane (cf. figure 6.6) and the following theorem gives us an upper bound for the J-value within this rectangle.

Theorem 15 *Given that*

$$\begin{aligned} Pr(P \sqsubseteq W) &\in [p_{min}, p_{max}] \\ Pr(R \sqsubseteq W) &\in [r_{min}, r_{max}] \end{aligned}$$

then the J-value of a rule $P \rightarrow R$ is given by

$$J(P \rightarrow R) = \begin{cases} J(p_{min}, r_{max}) & : \text{ if } p_{min} \geq r_{max} \\ J(p_{min}, p_{min}) & : \text{ if } p_{min} < r_{max} \wedge p^* < p_{min} \\ J(r_{max}, r_{max}) & : \text{ if } p_{min} < r_{max} \wedge p^* > r_{max} \\ J(p^*, p^*) & : \text{ if } p_{min} < r_{max} \wedge p_{min} \leq p^* \leq r_{max} \end{cases} \tag{6.8}$$

where $p^* = \frac{1}{2^{1/\ln 2}} \approx 0.37$ and

$$J : U \rightarrow \mathbb{R}, (y, x) \mapsto x \log\left(\frac{1}{y}\right) + (y - x) \log\left(\frac{y - x}{y(1 - x)}\right)$$

with $U = \{(y, x) \in [0, 1]^2 | y \geq x\}$ and the convention $0 \cdot \log 0 = 0$.

Proof of Theorem 15: The $J(y, x)$ term is nothing else than the J-measure, as we have seen in equation (6.4) of Theorem 13.

We have seen in Theorem 13 that J is monotonically increasing with the rule support $Pr(R \sqsubseteq W)$ abbreviated as $Pr(x)$. Let us now investigate the relationship to the premise support $Pr(P \sqsubseteq W)$ abbreviated as $Pr(y)$. Assume that $Pr(x)$ is fixed, then the partial derivative of J with respect to $Pr(y)$ is:

$$\begin{aligned} \frac{\partial J(X|Y=y)}{\partial Pr(y)} &= -\frac{Pr(x)}{Pr(y)} + 1 \cdot \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) + \\ &\quad (Pr(y) - Pr(x))\left(\frac{1}{Pr(y) - Pr(x)} - \frac{1 - Pr(x)}{Pr(y)(1 - Pr(x))}\right) \\ &= -\frac{Pr(x)}{Pr(y)} + \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) + 1 - \frac{Pr(y) - Pr(x)}{Pr(y)} \\ &= -\frac{Pr(x)}{Pr(y)} + \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) + 1 - \frac{Pr(y)}{Pr(y)} + \frac{Pr(x)}{Pr(y)} \\ &= \log\left(\frac{Pr(y) - Pr(x)}{Pr(y)(1 - Pr(x))}\right) \end{aligned}$$

From (3.8) we know that $Pr(y) > Pr(x)$. Since $Pr(y) \in [0,1]$ we have $Pr(y) - Pr(y) \cdot Pr(x) \geq Pr(y) - Pr(x)$ and thus $\frac{Pr(y)-Pr(x)}{Pr(y)(1-Pr(x))} \leq 1$. Therefore, the argument of the logarithm is always smaller than 1 and the logarithm is always negative. Thus, J is monotonically decreasing with the premise support $Pr(y)$.

Therefore the maximal J-value must be obtained either on the left border of the rectangle (if we move further to the right, that is, increase the support of the premise pattern, J decreases according to the above mentioned monotonicity) or on the top border (if we move downwards, that is, decrease the support of the rule pattern, J decreases according to Theorem 13). Consequently, if the top left element of the rectangle (p_{min}, r_{max}) is in the universe of discourse U (cf. figure 6.6(a)), that is $p_{min} \geq r_{max}$, the maximal value of J is obtained by $J(p_{min}, r_{max})$ which is the first case in (6.8).

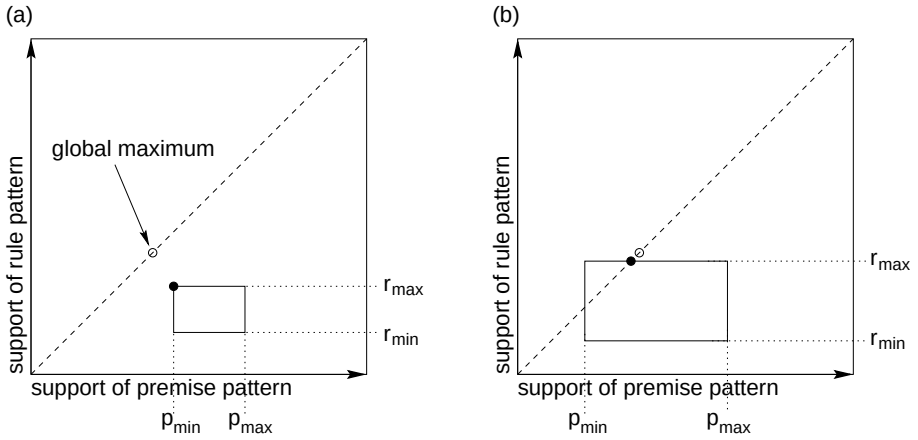


Figure 6.6: Estimating the maximal J-value within a rectangle of premise and rule pattern support.

For the other cases when $p_{min} < r_{max}$ the diagonal intersects the rectangle (cf. figure 6.6(b)). From the monotonicity we can conclude that the maximum J-value must lie on the diagonal. On the diagonal J reduces to $J(p,p) = p \cdot \log_2 \frac{1}{p}$, $p \in [0,1]$. We obtain a unique solution from

$$\frac{\partial J(p,p)}{\partial p} = \log_2 \frac{1}{p} - \frac{1}{\ln 2} = 0$$

at p^*. At p^* the J-value has a global maximum of ≈ 0.53. Thus, if (p^*, p^*) is contained in the rectangle, the maximal J-value is the global maximum of J (fourth case in (6.8)). Depending on whether the maximum is on the left or above the rectangle the maximum J-value is given by the intersection point of the diagonal with the left border or the top border of the rectangle (second and third case in (6.8)). ■

A revised algorithm is shown in figures 6.7 and 6.8. Compared to the algorithm in figure 6.4 we also abandoned the pure depth-first search, which explored the search space in a predetermined order regardless of the particular problem at hand. The new algorithm stores all reached nodes of the search tree in a map ordered by the maximally achievable J-value. (If the maximal J-values are identical for two nodes, they are ordered by the current J-value. If the current J-values are also identical, according to Occam's Razor [Mitchell, 1997] the node that uses fewer intervals should be preferred.) By selecting the next search node from this map

the search is guided to the most promising regions of the search space (A^* search [Pearl, 1984]). If memory space is exhausted (figure 6.7, line 9), we switch back to depth-first search with pruning (figure 6.8).

The revised algorithm as well as the brute force algorithm make use of an arbitrary order of the elements in $\mathcal{S}$ – just to make sure that the same subset is enumerated only once. To get most out of the node pruning, we sort the elements in $\mathcal{S}$ by their rule pattern support. For the estimate of the maximal J-value according to Theorem 15 the maximally achievable rule pattern support is of major importance: The larger the difference between the current rule pattern support and the maximally achievable rule pattern support, the more the maximally achievable J-value is usually overestimated, because the premise support increases in practice often faster than the rule support. By ordering the elements in $\mathcal{S}$ such that the rules with the highest rule support come first the overestimation is reduced compared to an arbitrary order. The tighter estimation reduces the average search depth needed until a node can be pruned. This is particularly important if the memory is exhausted and we have to switch back to depth-first search.

6.5 Evaluation

We want to consider an artificial test data set we have used earlier in section 4.3. We prefer to show some results using an artificially generated data set over a real data set, because only in this case do we *know* about the true patterns in the data. The following description of the data set is taken from page 92:

> We have generated a test data set where we have randomly switched three states A, B, and C on and off at discrete time points in $\{1, 2, ..., 9000\}$ with probability 0.2, yielding a sequence with 2838 states. Whenever we have encountered a situation where only A is active during the sequence generation, we generate with probability 0.3 a 4-pattern A meets B, B before C, and C overlaps a second B instance. The length and gaps in the pattern were chosen randomly out of $\{1, 2, 3\}$. [...] We consider the artificially embedded pattern and any subpattern consisting of at least 3 states as interesting.

Thus, the 3 rules $A \rightarrow BCD$, $AB \rightarrow CD$ and $ABC \rightarrow D$ were considered as interesting. In terms of episodes, this corresponds to a single interesting episode $ABCD$ (interval relationship omitted). For a window width of 16 and a very low minimum support threshold of 0.1% we obtain 17484 frequent patterns with up to 10 intervals. From the low support value, the comparatively small database, and the fact that we have only a single true relationship in the data, we expect many incidental patterns that are not meaningful. Considering only meaningful episodes rather than all patterns reduces the set to 8579 (49%), among them are 2425 maximal episodes (13.9%). However, we have found only 669 core episodes (3.8%), among them 515 maximal core episodes (2.9%). As expected, the best maximal core episode is our artifically embedded pattern:

$$A \rightarrow_{[0.0004,0.01]} B \rightarrow_{[0.36,0.46]} C \rightarrow_{[0.47,0.47]} B$$

The intervals denote the range of possible J-values for rules extracted from the episode. For instance, if the expert agrees with the episode and wants to use it as a rule, whatever conclusion he may select from the two last symbols, by selecting an appropriate subset from the premise he will obtain a J-value within $[0.36, 0.46]$.

```
1  proc find_best_subset(→ S[], ↔ bestJ, ← G[])
2    instantiate map of nodes nodes, ordered lexicogr. by (maxJ,J)
3    insert root node N with N.J = 0, N.maxJ = 0.53, N.idx = −1, N.inc ≡ 0
4    bestN ↤ N;                                  bestN is the best search node so far
5    while |nodes| > 0 do
6          pop first node N from nodes
7          oldbest ↤ bestJ;
8          if |nodes| > 200000                    threshold on maximal size of nodes
9            then                                 memory full, switch to depth-first search
10                copy N.inc to incl;
11                find_best_subset_dfs(N, N.idx, bestJ, incl);
12                if bestJ > oldbest
13                  then N.J ↤ N.maxJ ↤ bestJ; bestN ↤ N; fi
14             else                                memory available, A* search
15                for i ↤ N.idx + 1 to |S| do
16                    instantiate new node M;
17                    M.idx ↤ i;
18                    M.P ↤ N.P ∪ S[i].P;
19                    M.R ↤ N.R ∪ S[i].R;
20                    M.inc ↤ N.inc; M.inc[i] ↤ true;
21                    calculate N.J and N.maxJ according to Theorem 15;
22                    if M.J > bestJ
23                      then bestJ ↤ M.J; bestN ↤ M; insert M in nodes;
24                       else if M.maxJ > bestJ then insert M in nodes; fi
25                    fi
26                od
27         fi
28         if bestJ > oldbest
29           then remove last nodes in nodes with maxJ < bestJ
30         fi
31   od
32   for i ↤ 1 to |S| do
33       if bestN.inc[i] = true then G ↤ G ∪ S[i]; fi
34   od
35 .
```

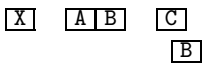
Figure 6.7: Revised specialization algorithm. A search node N consist of interval lists $N.P$ and $N.R$ for premise and rule pattern support, the current and maximal J-value in this branch $N.J$ and $N.maxJ$, a depth index $N.idx$, and a bit vector $N.inc$ indicating which elements of $\mathcal{S}$ have been considered in the search node N. The elements in $\mathcal{S}$ should be ordered decreasingly by their rule support.

```
1  proc find_best_subset_dfs(→ S[], ↔ N, → level, ↔ bestJ, ↔ include)
2    oldR ↤ R; oldP ↤ P;
3    for i ↤ level + 1 to |S| do
4        N.P ↤ oldP ∪ S[i].P;
5        N.R ↤ oldR ∪ S[i].R;
6        include[i] ↤ true;
7        calculate N.J and N.maxJ accord. to Theorem 15;
8        if N.J > bestJ
9          then bestJ ↤ N.J; N.inc ↤ include; fi
10       if N.maxJ > bestJ then find_best_subset_dfs(S, N, i, bestJ, include); fi
11       include[i] ↤ false;
12   od
13 .
```

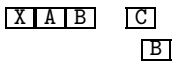
Figure 6.8: Subroutine used in figure 6.7 implementing a depth-first search with pruning.

The gap between the two rightmost J-intervals is smaller than the the gap between the two leftmost intervals, from which one can conclude that the premise "A" alone does not provide useful rules but in combination with B.

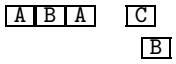
As already mentioned, the data set from section 4.3 was not designed to illustrate generalization. Therefore it is surprising to find a meaningful generalization of the $ABCD$ sequence. The following 11 superepisodes (depicted graphically) were considered during generalization:



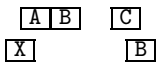
for $X \in \{A, B, C\}$, (6.9)



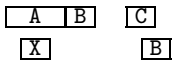
for $X \in \{B, C\}$, (6.10)



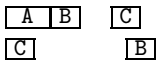
(6.11)



for $X \in \{B, C\}$, (6.12)



for $X \in \{B, C\}$, (6.13)



(6.14)

While a disjunctive combination of these specialized rules did not increase the J-value of the rule $ABC \rightarrow B$ it did for $AB \rightarrow CB$. The increase in the minimum J-value was only moderately (increase from 0.3618 to 0.3675), but nevertheless interesting to interpret. The generalization used 6 out of the possible 11 superepisodes. None of the episodes in (6.9) was used: For any meaningful pattern it is very likely to observe any of the labels A, B, or C in a *before* relationship if the window is sufficiently large. Consequently, these labels are not useful for predicting the continuation of the sequence and thus do not improve the J-value of a rule – it makes sense to discard these episodes during generalization. Also episode (6.11) was not considered: From the description how the patterns were generated, the second A instance tells us that the first 3 intervals were not generated according to the explanation given above – the pattern is thus incidental and correctly discarded. All other episodes, besides (6.12) for $X = B$ were used for generalization. While we

have no explanation why the case $X = B$ is excluded, the used episodes (6.10), (6.12), (6.13) and (6.14) share a common property: There is a B or C instance with non-empty intersection with the first A instance, but the right bound of the A instance is never included in this intersection. This summary comes pretty close to the original formulation

> Whenever we have encountered a situation where only A is active during the sequence generation, we generate [...]

With the considered pattern space we are not able to express that no other intervals besides A are observable at some point in time. However, if we observe that a B and/or C instance has ended just before an A instance is ended, it seems to be much more likely that the condition "only A is active" holds at the end of A. This observation increases the goodness-of-fit of the specialized rules but cannot be reflected by a single episode. Only by generalization we can find this relationship and overcome the inadequacy of the pattern space.

For a window width of $\Delta t_{win} = 18$ the specialization of the core episodes included the search for the best subset in rule sets of sizes 31, 25, 21, 20, etc. The algorithm found all best subsets within 4:30 minutes, the brute force algorithm was run for three hours and then aborted. The optimal subset for $|\mathcal{S}| = 31$ was determined within less than 30 seconds, whereas one case with $|\mathcal{S}| = 25$ was particularily time consuming (about 2 minutes). Thus, it is not primarily the size of $\mathcal{S}$, but the pecularities of the premise and rule support sets that influence the effectiveness of the pruning and thus the run-time of generalization.

Chapter 7

Application

In this chapter we discuss feature selection (labels for the interval sequence) and its impact on the mining process in terms of runtime and induced knowledge. We then apply the methodology developed in the previous chapters to the analysis of windspeed, air pressure, and wind direction data from a small island in the northern sea[1].

Feature Selection

There are, of course, no generally valid guidelines for feature selection, this step depends strongly on the application and available data, however, some general observations should be taken into consideration.

Variable Selection. With multivariate time series, we usually have to select a subset of the variables for the analysis to keep the computational burden tractable. Measures like the correlation coefficient can be used to identify redundant variables – if two time series behave very similar, one should better select only one variable from a set of similar or redundant variables for the analysis to reduce the complexity of the pattern mining. (On the other hand, sometimes it might be helpful to have redundant rules to be more robust against noise [Carrault et al., 2002].)

Label Refinement. For each variable, we may extract intervals for increasing/decreasing and/or convex/concave behaviour. If we do not discover discriminative relationships using increasing and decreasing segments only, rule specialization can be used to induce thresholds to refine the labels to "steeply increasing" or "shortly decreasing", for instance. But of course it is also possible to analyse the data beforehand to see if a further subdivision suggests itself. Cluster analysis can be applied to identify groups of attribute values that appear quite often. Clusters can be sought in univariate attributes, such as interval length (to distinguish long and short segments), as well as in multivariate attributes like the cross product of average first and second derivative, for instance.

However, we must be aware that a bad refinement may lead to an increase in the number of patterns without increasing the quality of the rules. As an example for a bad choice, consider that we have a cluster of segment lengths with centroid l, that

[1] Helgoland, 54:11N 07:54O

is, there are many intervals that have a length of nearly l. If we use this l to refine a label `inc` (denoting an increasing trend) to `inc-short` and `inc-long` (and l is not really relevant for discrimination), then probably every frequent pattern using `inc` will appear afterwards with an `inc-short` and `inc-long` variant, thereby doubling the number of patterns. Instead of choosing the cluster centre l we should better select a threshold that can discriminate between different clusters of interval lengths, that is, if there is another cluster at m of the same size the threshold $\frac{l+m}{2}$ would be better suited as a threshold, because it can distinguish segment lengths from both clusters. However, the existence of clusters does not necessarily mean that these lengths have any special meaning and lead to improved rules.

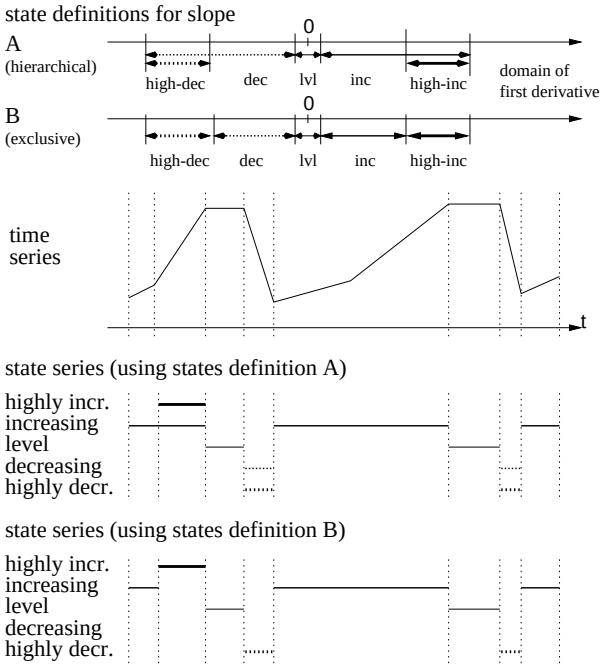


Figure 7.1: Two ways of partitioning a time series.

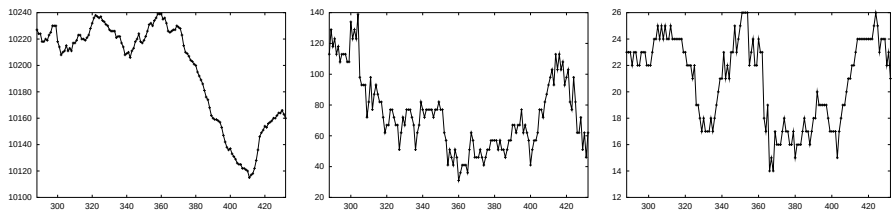
Label Taxonomy. In the previous paragraph we have discussed how clustering can be used to define thresholds for label refinement. For instance, among the increasing segments we may want to distinguish between those whose derivative is larger than l (steeply increasing) and m (very steeply increasing), $l < m$. There are two possibilities, either we define an exclusive partition ("increasing" segments have a slope within $[0, l[$, "steeply increasing" within $[l, m[$, and "very steeply increasing" within $[m, \infty[$) or we define hierarchical labels ("increasing" segments have a slope within $[0, \infty[$, "steeply increasing" within $[l, \infty[$, and "very steeply increasing" within $[m, \infty[$). Having chosen the threshold values heuristically, we cannot be sure that we have chosen them meaningful (with respect to some patterns we want to discover). If we are not sure about the threshold values, a hierarchical definition is preferable over the exclusive. There is a pattern "highly increasing segment meets level segment meets highly decreasing segment" in the time series depicted in figure 7.1. If the threshold values for the derivatives have not been chosen appropriately, the increasing flank of the second wave will not be classified as "highly increasing". But if we use the interval sequence A in figure 7.1, we will at least discover the pattern "increasing segment meets level segment meets highly decreasing segment". If we choose the exclusive label definition B in figure 7.1, the depicted interval se-

quence contributes only partially to the support of both discussed patterns. Even if the thresholds have been obtained from specialization of a single rule, we cannot be sure that the threshold has a *general meaning* that justifies the introduction of a new label that will occur in many other rules, too. With a hierarchical label definition we can at least be sure that we will find a *similar pattern* in terms of the employed label hierarchy. On the other hand, the hierarchical definition of labels yields higher support values in general and thus more frequent patterns will be discovered.

Label Combination. With multivariate time series we face the question whether we should learn interdependencies on each variable and merge the results or consider all variables from the beginning. The latter increases the complexity of the approach (more temporal patterns to discover) and thus the former can become a must if the number of variables increases. In this case, episodes can be identified in the variable and a new label is introduced for each interesting episode (such as replacing a pattern "short-decrease meets short-high-increase meets short-high-decrease" for the QRS complex in ECG patterns by a single interval labeled "QRS"). This increases the number of labels, but usually these episodes are less frequent than the primitive segments.

Application to Weather Data

The weather data set was obtained from the Deutsche Wetterdienst (DWD). To get a first impression, figure 7.2 shows 6 days of air pressure, wind strength, and wind direction data. Air pressure is measured in $\frac{1}{10}$ hPa, windspeed in $\frac{1}{10}$ metres per second, and wind direction is measured in 36 pieces (0 corresponding to north, 18 to south, etc.). This data set has been selected, because it is known to have interdependencies between the variables, which makes it suited to test and validate the approach.



(a) air pressure (b) windspeed (c) wind direction

Figure 7.2: Six days of observations.

About 7.5 years of hourly observations have been transformed into sequences of labeled intervals using the multiscale method discussed in section 2.2.3. A minimum stability of 2 resp. 3 has been required for air pressure and windspeed resp. wind direction. The extraction of ambiguous features was enabled, the pattern discovery was restricted to loosely connected patterns (cf. section 5.2). Thresholds were $\Delta t_{win} = 48h$, $\min_{supp} = 2\%$, $\min_{conf} = 25\%$.

Let us concentrate on forecasting strong winds, that is, windspeed of 5 Beaufort and above ($\geq 9.8\frac{m}{s}$). So far, we have only discussed how to segment a time series

into increasing and decreasing segments, but not how to extract an interval such that the signal is above a certain threshold τ within this interval: If the windspeed is sufficiently high it is very likely that noise make the windspeed profile oscillate around the landmark of 5 Beaufort. How can we extract a `w-strong` feature robustly from such a signal? The problem is related to the problem discussed in chapter 2. If we consider a transformed signal

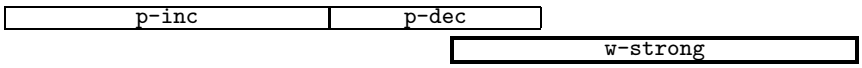
$$\int_t f(t) - \tau \, dt$$

of the original signal $f(t)$ then its derivative is $f(t) - \tau$ and zero-crossings correspond to regions where the original signal is below or above the threshold τ.

For a first step, we analyse `p-inc`/`p-dec` (increasing and decreasing wind strength), `d-inc`/`d-dec` (clockwise and counter-clockwise turning wind direction), `w-inc`/`w-dec` (increasing and decreasing windspeed) and `w-strong` labels only. Since the discovered rules are symbolic in nature, simple text matching algorithms (regular expression matching) can be applied to select a subset of rules that is of special interest. We extract those that contain `w-strong` in the conclusion and at least one interval that does not refer to the windspeed in the premise. This set of rules is specialized with respect to quantitative attributes such as the interval length, scale information, and maximal slope value (moving 3-point average).

The specialized rules that contained intervals referring to the windspeed itself, usually a `w-inc` segment, were not very useful since the constraints on the interval length (e.g. `w-inc` for more than $19h$) and slope (e.g. slope of `w-inc` greater then $0.36\frac{m}{s}$ per hour) correspond to a considerable increase in the windspeed (here $6.9\frac{m}{s}$), which is close to how `w-strong` was defined. More interesting are those patterns that refer only to wind direction and air pressure in the premise. In the following graphical rule representations the intervals in the conclusion are drawn with a thick border. The five top-ranking rules in this subset[2] are

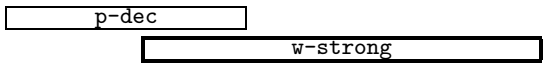
- The best rule predicts strong winds given a peak in the air pressure curve.



  A minimum segment lengh of $5h$ and $18h$ is required for the `p-inc` and `p-dec` segments. (There is also a constraint that the length of `p-dec` must be smaller than $43h$ but we assume that this constraint is related to the window width of $48h$: if this segment is much longer, the `p-inc` and `w-strong` segments cannot be observed in the same window.) The (3-point moving average) maximum slope of the `p-inc` segment has to be greater than $0.167\frac{hPa}{h}$ and for the `p-dec` segment smaller than -0.971$\frac{hPa}{h}$. A last constraint says that the `p-dec` segment must be observable on the finest scale. This rule allows good prediction with a confidence of 82%, specialization raised the J-value from initially 0.008bit to 0.17bit.

  We note in passing that rule #7 is almost identical, but the `p-dec` segment is extracted from a coarser scale and *contains* rather than *overlaps* the `w-strong` segment.

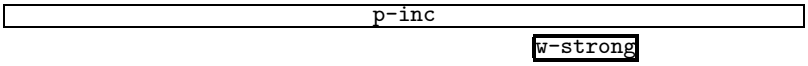
- The second best rule stems from the same maximal core episode, only the `p-inc` segment is missing.



[2] In the whole set of rules, the top ranking rules are dominated by deterministic relationships such as `w-inc` *meets* `w-dec` *meets* `w-inc`...

The same constraints on slope and length of the `p-dec` interval have been derived, but the missing constraints on the `p-dec` segment reduces the confidence of the rule to 59% (J=0.16 bit).

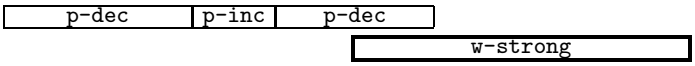
- Rule #3 looks like this:



This appears a bit contradictory to the first rules, especially rule #2, where steeply decreasing airpressure has to precede the strong winds. The constraints reveal the important information that the `p-inc` segment must be extracted from quite a coarse scale (≥ 4). The coarser the scale gets, the less local information is contained in the extracted intervals, therefore this rule is not very specific for `w-strong` segments. Since rule #6 is the same as #3 but `p-inc` is replaced by `p-dec`, the only conclusion we may derive is that strong winds are better predicted (or caused) by local phenomena (patterns in finer scales) rather than more global ones (patterns in coarser scales).

- Rule #4 is similar to #3, but `p-inc` is substituted by `d-inc`. The same argumentation as before applies.

- Rule #5 is a third variation of rule #1 and #2:



The rule confidence is 61% (J=0.13 bit) and the constraint on the slope of the second `p-dec` segment is relaxed to $\leq -0.66\frac{hPa}{h}$. The slope of the first `p-dec` segment must be larger than $-1.06\frac{hPa}{h}$ (otherwise the `w-strong` pattern will occur earlier as in rules #1 and #2).

Rules #1, #2 and #5 emphasize a strong relationship between decreasing air pressure and strong winds. This is in correspondence to the literature [Karnetzki, 1999, Sprecher Energie, 1990] and an indication for the validity of the results obtained by the method. We also revealed an important threshold (slope of `p-dec` must be smaller than -0.971$\frac{hPa}{h}$), which is pretty close to the threshold of $-1.0\frac{hPa}{h}$, which is mentioned in [Sprecher Energie, 1990].

We also analyzed patterns with `p-ccv`/`p-cvx` (concave and convex segments) instead of `p-inc`/`p-dec` segments. Most of the rules with `p-inc`/`p-dec` segments can be formulated by means of concave and convex behaviour, however, the J-value and confidence of the `p-inc`/`p-dec`-rules was higher, therefore we skip the examples.

The investigations in the discovered rules lead us to a revision of the set of labels. We generate new intervals that reflect what we have found so far:

- We introduce a `p-vsteep` label for a very steep air pressure curve (absolute value of maximum slope $\geq 0.97\frac{hPa}{h}$, rule #1). A number of rules use a threshold $\approx 0.6\frac{hPa}{h}$, which is used to define a `p-steep` label. A symbol for a slowly changing air pressure curve with maximum slope $\leq 0.1\frac{hPa}{h}$ is also extracted (`p-flat`).

- In the `p-ccv`/`p-cvx` experiment many rules used a threshold of ± 0.04 for a maximum in the second derivative. We use this threshold to introduce `p-bent-ccv`/`p-bent-cvx` labels.

- The inspection of the `w-inc`/`w-dec`-rules reminded us of the fact that steep windspeed curves are good indicators for upcoming strong winds, therefore we use the discovered thresholds $\approx 0.6\frac{m}{s\,h}$ to introduce `w-steep-inc`/`w-steep-dec` labels. A label `w-storm` for 7 Beaufort windspeed ($\geq 15.2\frac{m}{s}$) is also included.

- The threshold of $0.2\frac{deg}{h}$ occurred with slight variations in other rules, therefore we introduce appropriate `d-steep-inc`/`d-steep-dec`-labels (and removed the old `d-inc`/`d-dec` intervals). A threshold of $0.1\frac{deg}{h}$ on the maximum gradient is used to define `p-stable` segments (constant wind direction).

- The observation that local phenomena are more important is taken into account by extracting no intervals that start at scale 4 or above.

This dictionary of 17 different labels lead us to a sequence of 30209 intervals for the ≈ 7.5 years of observations. Figure 7.3 summarizes the pattern mining step. Less than 1% of the frequent patterns were maximal core episodes.

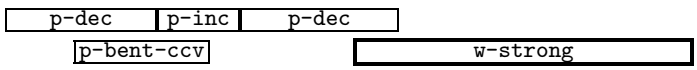
k	$\lvert F_k\rvert$	$\lvert F_k\rvert/\lvert C_k\rvert$	$\lvert C_k\rvert$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	17	100%	17	0	0	0	1.22	0
2	570	35.7%	1598	0	0	0	52.25	0.03
3	5878	11.3%	51970	96997	19982	95191	138.12	4.87
4	23557	23%	102470	1581159	782479	336301	265.44	38.86
5	46249	82.8%	55889	5184846	3086753	198878	255.94	102.21
6	43734	94.9%	46072	7025226	4547977	141095	261.88	132.37
7	24253	96.9%	25024	4375758	3223071	42808	192.7	79.57
8	9382	96.5%	9723	1649632	1298912	13753	102.01	33.36
9	2566	95.5%	2687	470238	358129	4296	44.71	9.62
10	365	94.1%	388	91484	61614	773	15.06	2.03
11	16	100%	16	7178	5243	50	6.58	0.14
12	0	0%	0	174	273	0	0.02	0
$\sum$	156587	52.9%	0.8%	58.5%	38.2%	2.5%	1738.99	

Figure 7.3: Experiment with enriched interval sequence.

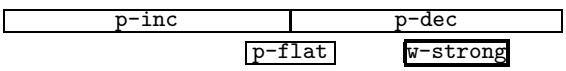
As before we restrict ourselves to rules that have a windspeed attribute in the conclusion and additionally require that no windspeed attribute is in the premise pattern (to avoid rules of the kind "`w-steep-inc`→ `w-strong`"). The best rule after specialization now concludes from a "`p-inc` *meets* `p-dec`" pattern to `w-steep-inc` (J=0.23 bit) rather than `w-strong`.

Let us have a closer look at some of the newly discovered rules:

- Rule #5 of the previous experiment is extended by a `p-bent-ccv` label, which increases the rule confidence from 61% to 85% (and decreases the support from 0.04% to 0.03%, J-value almost unchanged).

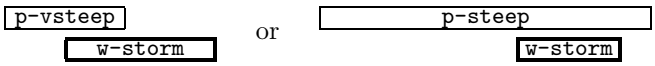


- In a variant of rule #1 a `p-flat` label describes the local maximum in more detail (not a sharp peak but a plateau):



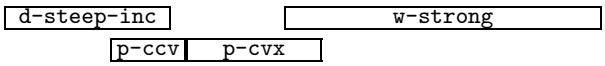
The minimum duration for the `p-inc`, `p-dec`, and `p-flat` interval are $4h$, $3h$, and $17h$, resp. The slope of the `p-inc` and `p-dec` segments must be less than $1.15\frac{hPa}{h}$ and $-0.47\frac{hPa}{h}$, resp. The confidence is 68% and $J = 0.10$ bit.

- As expected from rules #1, #2 and #5, the `p-steep` attribute alone does not discriminate between strongly increasing or decreasing windspeed, we obtain rules "`p-steep` → `w-steep-inc`" and "`p-steep` → `w-steep-dec`" (rule that contain additional `p-inc`/`p-dec` segments are more precise). However, all rules that conclude to `w-storm` segments contain a `p-steep` or `p-vsteep` segment in the premise, for instance



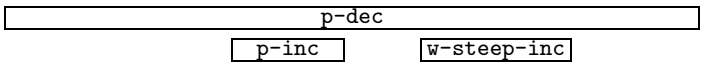
For the first rule the `p-vsteep` segment must be $6h$ to $21h$ long and the average (absolute value of the) gradient must be larger than $1.35\frac{hPa}{h}$. For the second rule, the `p-steep` segment must be longer than $21h$. While the confidence in the first rule is only 35%, it is 50% for the second rule, which is not bad for a comparatively seldom event like storm.

- Compared to rule #4 in the first run, there are more reliable rules using features of the wind direction curve:

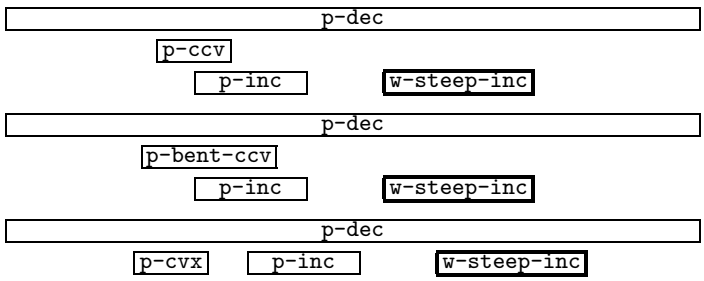


While the support of this rule is relatively low (2%) its confidence is quite good (71%, J=0.08 bit). The constraints require a minimum length of $10h$ for the `p-cvx` segment and a maximum value smaller than -0.059 for the second derivative.

- While coarse features alone were not very helpful for prediction, they can be useful in combination with detailed features to get a better impression of the overall outlook of a curve. The following episode is an example of such a case:

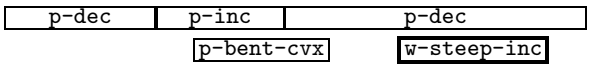


Compared to rule #3 in the first run, the confidence has increased from 50% to 62%, mainly because the `p-dec` segment stems from a scale ≤ 3. Generalizing the core episode leads to the following disjunctive combination



All three episodes describe the existence of a local minimum within the global downward trend and the generalization to a single episode is therefore justified. This is one of the situations where we have ambiguity in the perception of the air pressure curve, and from the relationships between `p-inc`/`p-dec` and `w-dec`/`w-inc` we may want to conclude that in this case the more global trend is dominating (if the average `p-dec` gradient is smaller than $-0.093\frac{hPa}{h}$ and the `p-dec` interval is longer than $55h$).

- Finally, we want to give a few examples for the actual signals the rules are referring to. Figures 7.4 to 7.7 show sliding windows that contain instances of the rule



(7.1)

  with a minimum length of $21h$ for the last `p-dec` segment (and various omitted constraints on the slope values of the `p-inc`/`p-dec` segments).

  Each figure shows the air pressure curve and the associated intervals in the upper half and the windspeed in the lower. For the air pressure the row denoted "bent" contains the `p-bent-ccv`/`p-bent-cvx` segments, "steep" contains the `p-steep` segments, and "grad" contains the `p-inc`/`p-dec` segments. For the windspeed intervals the row denoted "steep" contains the `w-steep-inc`/`w-steep-dec` intervals, "strong" contains the `w-strong` intervals, and "grad" the `w-inc`/`w-dec` segments. All features are multiscale features, scales 0-6 are shown for gradient information, scales 0-2 for all others.

  The overall appearance of the sliding windows is quite different, a pointwise distance measure would yield high distance values for the four examples. However, one can easily detect a local minimum followed by a longer decreasing segment (`p-dec` `p-inc` `p-dec` sequence). The `p-bent-cvx` interval makes sure that we have sufficiently high curvature at the beginning of the long decreasing segment. In the conclusion we have a steep increase in the windspeed during the final `p-dec` segment.

Most of the rule patterns of the presented rules belong to maximal core episodes. Unfortunately we were not able to verify for all of the remaining rules if they are part of a generalized core episode, since we were not able to calculate the best generalization in all cases due to a large number of superepisodes. Thus, there is still room for improving the generalization step. Alternatively, in an interactive episode browser the user could select a subset of the superepisodes she or he wants to consider during generalization, thereby by-passing the computational limitations.

Further Experiments

At the beginning of this section we have discussed the possibility of an a priori label refinement on the basis of cluster analysis. Figure 7.8 shows a histogram of the obtained interval lengths, grouped by labels `p-inc/p-dec`, `p-ccv/p-cvx`, `w-inc/w-dec` and `d-inc/d-dec`. The first two graphs both show a maximum at a duration of 5 hours and then decrease monotonically. Due to the comparatively noisy windspeed and wind direction signals (cf. figure 7.2), we probably would have obtained a monotonically decreasing curve without a maximum at 5 hours if not a minimum stability of 2 was required during interval extraction from the tree of scales.

The last two graphs of figure 7.8, however, unveil an interesting periodicity: The extracted interval lengths for air pressure slope or curvature have local minima at multiples of 12 hours. Since thresholds of $12h$, $24h$, etc. are well-suited to discriminate these peaks from each other, we have refined the air pressure features according to this observation. However, the results were not satisfying, the refined labels occured seldomly in the rules. As we have already seen in the discussion, most of the thresholds obtained from rule specialization differed from multiples of $12h$. This underlines that thresholds obtained from cluster analysis are not necessarily

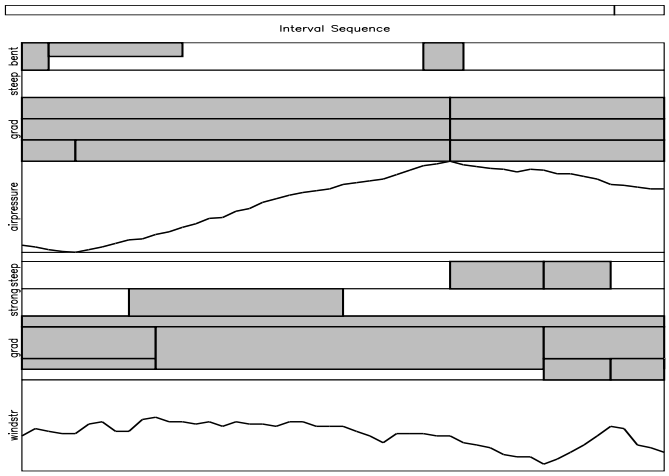


Figure 7.4: An instance of the pattern (7.1).

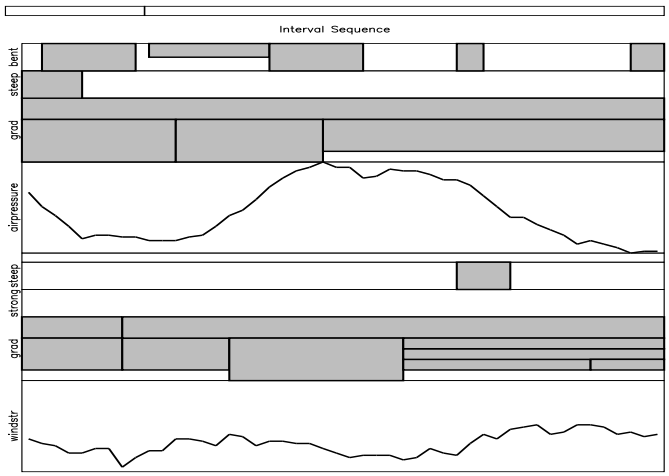


Figure 7.5: An instance of the pattern (7.1).

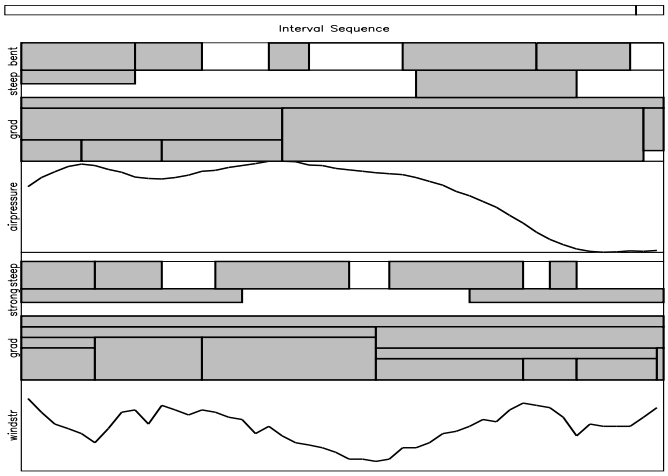


Figure 7.6: An instance of the pattern (7.1).

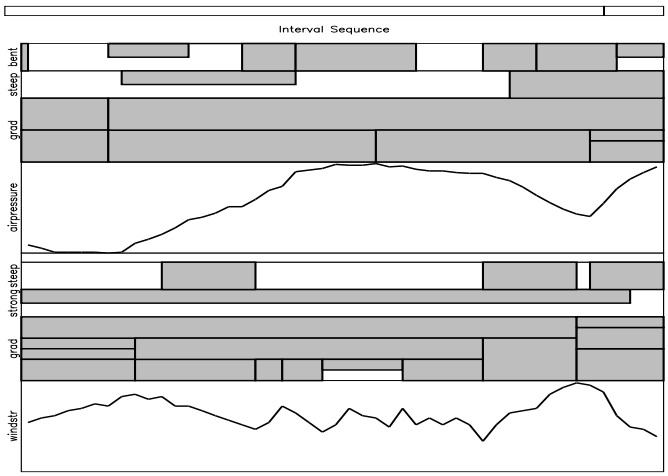
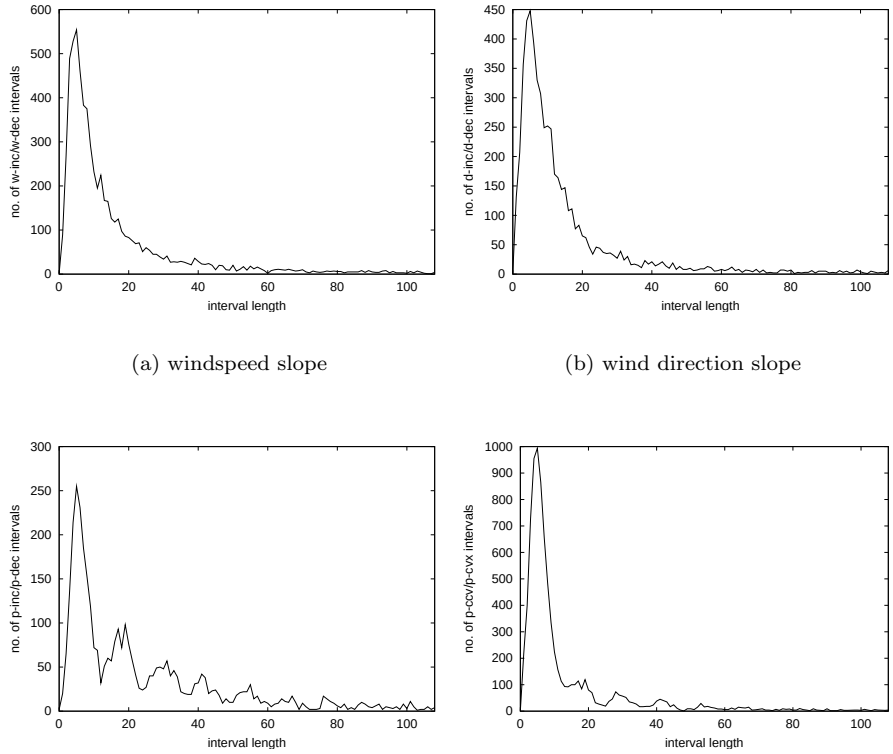


Figure 7.7: An instance of the pattern (7.1).



(a) windspeed slope

(b) wind direction slope

(c) air pressure slope

(d) air pressure curvature

Figure 7.8: Histograms of interval segment lengths.

meaningful for prediction (or classification) purposes. (If the histogram is further refined with respect to the maximum slope of the segment, a periodicity of $6h$ is revealed, which is caused by the turn of the tides.)

Experiments with the representation discussed in section 2.1.1.2 were also conducted. Any three consecutive `p-inc`/`p-dec` intervals (meeting each other) from the tree of scales were combined and the interval lengths and average slopes were used to form a 6-dimensional feature vector. This data set has been partitioned into 40 clusters using the fuzzy c-means algorithm. The same was done with the windspeed curve. For the final interval sequence the labels `w-storm`, `w-strong` and `w-steep-inc` as described earlier were also used, 22610 intervals in total. The data mining step is summarized in figure 7.9. About 63% of the frequent patterns were maximal core episodes.

The kind of patterns we can describe using this approach is of course limited compared to the features we have extracted before. In the set of rules and episodes we have not found any relationships leading from air pressure to `w-storm` or `w-strong` segments. The results from specialization are also less helpful, since the labels partition the set of shapes exclusively rather than hierarchically and thus the thresholds are not globally meaningful but refer explicitly to the shapes in the cluster under consideration.

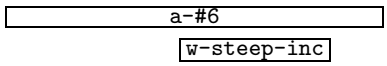
However, there is one advantage of this representation, which is that there are very few interdependencies between labels of the same signal, whereas a huge number

k	$\|F_k\|$	$\|F_k\|/\|C_k\|$	$\|C_k\|$	p_1	p_2	p_3	t_{SE}	t_{CG}
1	78	100%	78	0	0	0	0.61	0
2	3276	9.8%	33501	0	0	0	99.75	0.48
3	1161	1.5%	79086	893693	279805	1340133	41.39	17
4	131	26.8%	488	107898	55891	5806	4.54	0.56
5	7	87.5%	8	4648	3200	63	3.39	0.02
6	0	0%	0	146	70	0	0	0
$\sum$	4653	4.1%	4%	35.9%	12.1%	48%	167.74	

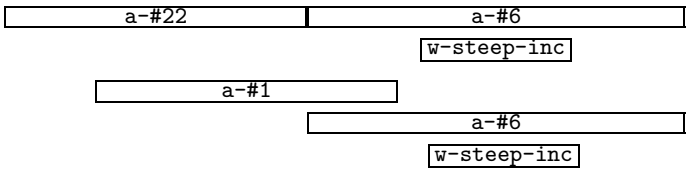
Figure 7.9: Experiment with an interval sequence obtained from clustering embedded models.

of patterns has been found before referring to known interval relationships induced from the tree of scales (like "`p-dec` *starts* `p-dec`", etc.). This makes the pattern mining step much faster and the data volume much smaller. As a positive side-effect, the number of superepisodes that has to be considered during specialization is also reduced.

Regarding `w-steep-inc` intervals, the first maximal core episode that gives us a relationship between `w-steep-inc` and airpressure is



It denotes a relationship between cluster #6 of the air pressure shapes and a contained `w-steep-inc`. The generalization of this core episode used two more labels referring to the air pressure:



The prototypical shapes of the air pressure clusters are

a-#1		a-#6		a-#22	
duration	average slope	duration	average slope	duration	average slope
10	-0.13	23	-0.64	8	0.10
10	0.12	22	0.47	8	-0.09
26	-0.54	23	-0.59	32	0.29

To figure out which range of possible interval lengths and gradients are characteristic for this rule, one has to take the remaining clusters into account (that is, the user has to be aware of the used codebook, which makes interpretation a bit more difficult than before). There are possible relationships to the patterns we have discovered before, but a direct comparison is difficult since we do not know the temporal relationship of the `w-steep-inc` segment to the three subsegments in the `a-*` patterns. From the overlap of `a-#1` and `a-#6` we can conclude that the `w-steep-inc` segment is located somewhere in the peak at the end of `a-#6`, which establishes a relationship to some other rules we have found before (for instance rule #1 of the first experiment). In general, we find the pattern representation we have used before easier to understand, but as mentioned at the beginning of this section, switching to this kind of label extraction can be advantageous if many time series have to be examined simultaneously.

Other areas of application can be found in bioinformatics, medical diagnosis, or even music. A score can be considered as an interval sequence, where the label denotes the tone and the interval the period of time when the tone shall be played. The algorithm was applied to a set of chorales by J.S. Bach which was obtained from the UCI machine learning repository [Blake and Merz, 1998]. The data set is univariate in the sense that it contained only a single voice out of originally four voices. Here are two rules that were discovered[3]:

G.4	H.4	D.5	**H.4**

When we play a (G major) chord in an arpeggio style upwards, then we continue playing it downwards.

D.5	H.4	G#.4	**A.4**

When we play an E7 chord in an arpeggio style downwards, then we end with the note A. (A typical cadence in A is finished by the chords E or E7 followed by A.)

[3]Many thanks to Frank Klawonn for providing the interpretation of the rules.

Conclusion

Knowledge discovery is learning from examples in the past. A human is very good in recognizing repeating situations and developing hypotheses about dependencies in the data, even if only moderate background knowledge is present. We have seen that temporal data is often subject to dilatation and translation effects and that a human is capable of recognizing the similarity of the situations despite of these effects. This fact has been realized before and techniques like dynamic time warping have been developed to compensate them. But a human perceives a time series visually and matches portions of the time series via some simple shapes rather than employing some sort of time warping. Thus, matching of time series can be considered as matching segments of primitive shapes, which is the central idea of this work.

The advantages of this approach are manifold:

- Similar to the human perception the used pattern representation is to some extent invariant to dilatation and translation effects, which makes it very likely that situations that are perceived as similar by a human show up as frequent patterns of the pattern space.
- Compared to techniques like dynamic time warping, pattern matching becomes much easier and thus more efficient.
- Multivariate time series and singular events are easily incorporated in this notion of temporal patterns.
- While most approaches require that the time series have to be similar in a fixed period of time (given by some sliding window), only partial similarity (partial matching) is considered in this approach (which makes the choice of the window width less critical).
- Last but not least, representing a time series as a sequence of labeled intervals reduces the data volume and thereby reduces the complexity of consecutive steps.

We have described a thorough realization of this approach, which consists of the following steps: time series abstraction, pattern mining, rule specialization, episode generalization, and ranking of discovered patterns. For the abstraction and ranking problems we have identified outstanding solutions in the literature (wavelet tree of scales, J-measure) and have shown that the vast majority of competive abstraction methods is not suited for our needs. For the other problems of pattern mining in a stream of labeled intervals, specialization of rules and generalization of episodes, we have developed completely new solutions that are explicitly tailored to interval sequence data. Most of the competetive approaches that have been found in the literature either use techniques originally designed for static data and thus cannot

reflect the pecularities of temporal data such as dilatation or translation, or use significant amount of background knowledge rather than being universally applicable to temporal/sequential data.

Most of the terminology and aspects addressed by humans when argueing about sequential data can be handled by the developed set of methods. If the methods are applied in the same way as a human analyzes temporal data, they are considered as sufficiently efficient. Since a human is used to the iterative refinement of concepts, the first step is to look for qualitative patterns in a coarse resolution without too many details. This pattern mining step is linear in the length of the sequence and log-linear in the number of discovered relationships. Once a human has selected potentially useful patterns, a more detailed view on the data is chosen and more details are considered. The proposed specialization step provides evidence for the introduction of thresholds on interval length, gradient information or whatsoever, that is, to refine the concepts that have been used so far. This step is more complex (in the worst case quadratic in the number of examples), but executed for a much smaller dataset. The concept of maximal core episodes can be used to present the discovered patterns in a condensed representation to an expert. Episode generalization helps to overcome the search bias any machine learning algorithm must have and can be used after having focussed on a small set of interesting findings.

Knowledge discovery is always an interactive process. Even the best heuristics to find "interesting" patterns cannot substitute the knowledge and intuition of a human. Therefore the analyst benefits from the fact that she or he can identify the discovered patterns immediately in the raw data by simple visual inspection. This makes interaction with the system particularily simple, since new hypothesis can easily be tested by adding the newly invented features into the stream of intervals.

Another important advantage of this work, that is missing in almost all other approaches, is the ambiguity inherent in the visual perception of time series. It is difficult to tell a priori, whether a subsegment of a time series should be considered as an increasing segment or a sequence of increasing-decreasing-increasing segments. Usually, the attempt to get a clear answer to such questions from an expert of the field fails awfully – "*it depends on the context*" is what we hear at best. But this answer characterizes the problem exactly, depending on what we are looking for a small bump in the series may be relevant or not. The problem is that in knowledge discovery we do know in advance what we are looking for, and thus taking this ambiguity into account during the discovery process is of vital importance.

The examples and applications have shown that the proposed framework is indeed capable of discovering relevant relationship of reasonable complexity in multivariate temporal data.

Bibliography

[Agrawal et al., 1993] Agrawal, R., Faloutsos, C., and Swami, A. (1993). Efficient similarity search in sequence databases. In *Proc. of the 4th Int. Conf. on Foundations of Data Organizations and Algorithms*, pages 69–84, Chicago.

[Agrawal et al., 1995] Agrawal, R., Lin, K.-L., Sawhney, H. S., and Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. of the 21st Int. Conf. on Very Large Databases*, Zürich, Switzerland.

[Agrawal et al., 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1996). Fast discovery of association rules. In [Fayyad et al., 1996], chapter 12, pages 307–328. MIT Press.

[Ahonen-Myka, 1999] Ahonen-Myka, H. (1999). Finding all maximal frequent sequences in text. In Mladenic, D. and Grobelnik, M., editors, *Proc. of the ICML99 Workshop on Machine Learning in Text Data Analysis*, pages 11–17.

[Allen, 1983] Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Comm. ACM*, 26(11):832–843.

[Attneave, 1954] Attneave, F. (1954). Some informational aspects of visual perception. *Psychol. Rev.*, 61:183–193.

[Babaud et al., 1986] Babaud, J., Witkin, A. P., Baudin, M., and Duda, R. O. (1986). Uniqueness of the Gaussian kernel for scale-space filtering. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(1):26–33.

[Bakshi, 1999] Bakshi, B. R. (1999). Multiscale analysis and modeling using wavelets. *Journal of Chemometrics*, 13:415–434.

[Bakshi and Stephanopoulos, 1995] Bakshi, B. R. and Stephanopoulos, G. (1995). Reasoning in time: Modelling, analysis, and pattern recognition of temporal process trends. In *Advances in Chemical Engineering*, volume 22, pages 485–548. Academic Press, Inc.

[Bellazzi et al., 2002] Bellazzi, R., Larizza, C., Magni, P., and Bellazzi, R. (2002). Quality assessment of dialysis services through intelligent data analysis and temporal data mining. In *Proc. of the ECAI'02 Workshop on Knowledge Discovery from (Spatio-) Temporal Data*, pages 3–9, Lyon, France.

[Berndt and Clifford, 1996] Berndt, D. J. and Clifford, J. (1996). Finding patterns in time series: A dynamic programming approach. In [Fayyad et al., 1996], chapter 9, pages 229–248. MIT Press.

[Berthold and Hand, 1999] Berthold, M. and Hand, D. J., editors (1999). *Intelligent Data Analysis*. Springer.

[Bezdek, 1980] Bezdek, J. C. (1980). A convergence theorem for the fuzzy ISO-DATA clustering algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2(1):1–8.

[Blake and Merz, 1998] Blake, C. L. and Merz, C. J. (1998). UCI repository of machine learning databases.

[Bradley et al., 1999] Bradley, P. S., Fayyad, U. M., and Reina, C. A. (1999). Scaling EM (expectation-maximization) clustering to large databases. Technical Report 15, Microsoft.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, Wadsworth, CA.

[Cãlin and Gâlea, 2001] Cãlin, M. and Gâlea, D. (2001). A fuzzy relation for comparing intervals. In Reusch, B., editor, *Proceedings of the 7th Fuzzy Days*, volume 2206 of *LNCS*, pages 904–916, Dortmund, Germany. Springer.

[Capelo et al., 1998] Capelo, A. C., Ironi, L., and Tentoni, S. (1998). Automated mathematical modelling from experimental data: An application to material science. *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, 28(3):356–370.

[Carrault et al., 2002] Carrault, G., Cordier, M.-O., Quiniou, R., and Wang, F. (2002). Intelligent multichannel cardiac data analysis for diagnosis and monitoring. In *Proc. of the ECAI'02 Workshop on Knowledge Discovery from (Spatio-) Temporal Data*, pages 10–16, Lyon, France.

[Chatfield, 1989] Chatfield, C. (1989). *The Analysis of Time Series – An Introduction*. Chapman and Hall, 4th edition.

[Chen et al., 1998] Chen, J.-Q., Xi, Y.-G., and Zhang, Z.-J. (1998). A clustering algorithm for fuzzy model identification. *Fuzzy Sets and Systems*, 98:319–329.

[Cherkassky and Mulier, 1998] Cherkassky, V. S. and Mulier, F. (1998). *Learning from Data*. Wiley.

[Chui, 1992] Chui, C. K. (1992). *An Introduction to Wavelets*, volume 1 of *Wavelet Analysis and its Applications*. Academic Press, Inc.

[Chung et al., 2001] Chung, F.-l., Fu, T.-C., Luk, R. W. P., and Ng, V. (2001). Flexible time series pattern matching based on perceptually important points. In *IJCAI-01 Workshop on Learning from Temporal and Spatial Data*, pages 1–7, Seattle, USA.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:262–283.

[Cohen, 2001] Cohen, P. R. (2001). Fluent learning: Elucidating the structure of episodes. In *Proc. of the 4th Int. Symp. on Intelligent Data Analysis*, number 2189 in LNAI, pages 268–277. Springer.

[Cohen and Adams, 2001] Cohen, P. R. and Adams, N. (2001). An algorithm for segmenting categorical time series into meaningful episodes. In *Proc. of the 4th Int. Symp. on Intelligent Data Analysis*, number 2189 in LNAI, pages 197–205. Springer.

[Das et al., 1998] Das, G., Lin, K.-I., Mannila, H., Renganathan, G., and Smyth, P. (1998). Rule discovery from time series. In *Proc. of the 4th Int. Conf. on Knowl. Discovery and Data Mining*, pages 16–22. AAAI Press.

[de Boor, 1978] de Boor, C. (1978). *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Science*. Springer.

[de Boor and Rice, 1968a] de Boor, C. and Rice, J. R. (1968a). Least squares cubic spline approximation I – fixed knots. Technical Report CSD TR 20, Dept. of Computer Sciences, Purdue Univ.

[de Boor and Rice, 1968b] de Boor, C. and Rice, J. R. (1968b). Least squares cubic spline approximation II – variable knots. Technical Report CSD TR 21, Dept. of Computer Sciences, Purdue Univ.

[Duda and Hart, 1973] Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York.

[Dunn, 1973] Dunn, J. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact, well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xiaowei, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases*, pages 226–331, Portland, Oregon.

[Falkenhainer and Forbus, 1991] Falkenhainer, B. and Forbus, K. D. (1991). Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95–143.

[Faloutsos et al., 1994] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *Proc. of ACM SIGMOD Int. Conf. on Data Management*.

[Fayyad et al., 1996] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996). *Advances in Knowledge Discovery and Data Mining*. MIT Press.

[Flexer, 1999] Flexer, A. (1999). On the use of self-organizing maps for clustering and visualization. In *Proc. of the 3rd Europ. Conf. on Principles of Data Mining and Knowl. Discovery*, volume 1704 of *LNAI*, pages 80–88, Prague, Czech Republic. Springer.

[Forbus, 1981] Forbus, K. D. (1981). Qualitative reasoning about physical processes. In *Proc. of the 7th Int. Joint Conf. on Artifial Intelligence*, pages 326–330.

[Forbus, 1987] Forbus, K. D. (1987). Interpreting observations of physical systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 17(3):350–359.

[Friedman, 1991] Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Stat.*, 19:1–141.

[Gaffney and Smyth, 1999] Gaffney, S. and Smyth, P. (1999). Trajectory clustering with mixtures of regression models. UCI-ICS 99-15, Dept. of Information and Computer Science, Univ. of California, Irvine.

[Geva, 1999] Geva, A. B. (1999). Non-stationary time-series prediction using fuzzy clustering. In Davé, R. N. and Sudkamp, T., editors, *Proc. of the 18th Int. Conf. of the North American Fuzzy Information Processing Society*, pages 413–417.

[Goodrich, 1994] Goodrich, M. T. (1994). Efficient piecewise-linear function approximation using the uniform metric. In *Proc. of the 10th Symp. on Computational Geometry (SCG)*, pages 322–331.

[Guimarães and Ultsch, 1999] Guimarães, G. and Ultsch, A. (1999). A method for temporal knowledge conversion. In Hand, D. J., Kok, J. N., and Berthold, M. R., editors, *Proc. of the 3rd Int. Symp. on Intelligent Data Analysis*, pages 369–380, Amsterdam, The Netherlands. Springer, Berlin.

[Han and Fu, 1995] Han, J. and Fu, Y. (1995). Discovery of multiple-level association rules from large databases. In *Proc. of the 21st Int. Conf. on Very Large Databases*, pages 420–431.

[Hathaway and Bezdek, 1993] Hathaway, R. J. and Bezdek, J. C. (1993). Switching regression models and fuzzy clustering. *IEEE Trans. on Fuzzy Systems*, 1(3):195–204.

[Holschneider et al., 1989] Holschneider, M., Kronland-Martinet, R., Morlet, J., and Tchamitchian, P. (1989). *Wavelets, Time-Frequency Methods and Phase Space*, chapter Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform, pages 289–297. Springer, Berlin.

[Honkela et al., 1997] Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. (1997). WEBSOM – self-organizing maps of document collections. In *Proc. of Workshop on Self-Organizing Maps (WSOM), June 4-6*, pages 310–315.

[Höppner, 2000] Höppner, F. (2000). Piecewise linear function approximation by alternating optimization. In *Proc. of the 8th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU)*, pages 1751–1757, Madrid, Spain.

[Höppner, 2001a] Höppner, F. (2001a). Discovery of temporal patterns – learning rules about the qualitative behaviour of time series. In *Proc. of the 5th Europ. Conf. on Principles of Data Mining and Knowl. Discovery*, number 2168 in LNAI, pages 192–203, Freiburg, Germany. Springer.

[Höppner, 2001b] Höppner, F. (2001b). Learning temporal rules from state sequences. In *Proc. of the IJCAI'01 Workshop on Learning from Temporal and Spatial Data*, pages 25–31, Seattle, USA.

[Höppner, 2002a] Höppner, F. (2002a). Discovery of core episodes from sequences – using generalization for defragmentation of rule sets. In *Pattern Detection and Discovery*, number 2447 in LNAI, pages 199–213. Springer.

[Höppner, 2002b] Höppner, F. (2002b). Handling feature ambiguity in knowledge discovery from time series. In *Proc. of 5th Int. Conf. on Discovery Science*, number 2534 in LNCS, pages 398–405, Lübeck, Germany. Springer.

[Höppner, 2002c] Höppner, F. (2002c). Learning dependencies in multivariate time series. In *Proc. of the ECAI'02 Workshop on Knowledge Discovery from (Spatio-) Temporal Data*, pages 25–31, Lyon, France.

[Höppner, 2002d] Höppner, F. (2002d). Speeding up fuzzy c-means: Using a hierarchical data organisation to control the precision of membership calculation. *Fuzzy Sets and Systems*, 128(3):365–378.

[Höppner, 2002e] Höppner, F. (2002e). Time series abstraction methods – a survey. In *Proceedings GI Jahrestagung Informatik, Workshop on Knowl. Discovery in Databases*, Lecture Notes in Informatics, pages 777–786, Dortmund, Germany.

[Höppner and Klawonn, 2000a] Höppner, F. and Klawonn, F. (2000a). Fuzzy clustering of sampled functions. In *Proc. of the 19th Int. Conf. of the North American Fuzzy Information Processing Society*, pages 251–255, Atlanta, USA.

[Höppner and Klawonn, 2000b] Höppner, F. and Klawonn, F. (2000b). Obtaining interpretable fuzzy models from fuzzy clustering and fuzzy regression. In *Proc. of the 4th Int. Conf. on Knowledge-Based Intelligent Engineering Systems & Allied Technologies*, pages 162–165, Brighton, UK.

[Höppner and Klawonn, 2001a] Höppner, F. and Klawonn, F. (2001a). Finding informative rules in interval sequences. In *Proc. of the 4th Int. Symp. on Intelligent Data Analysis*, volume 2189 of *LNCS*, pages 123–132, Lissabon, Portugal. Springer.

[Höppner and Klawonn, 2001b] Höppner, F. and Klawonn, F. (2001b). A new approach to fuzzy partitioning. In *Proc. of the Joint 9th IFSA World Congress and 20th NAFIPS Int. Conf.*, pages 1419–1424, Vancouver, Canada.

[Höppner and Klawonn, 2002a] Höppner, F. and Klawonn, F. (2002a). Finding informative rules in interval sequences. *Intelligent Data Analysis – An International Journal*, 6(3):237–256.

[Höppner and Klawonn, 2002b] Höppner, F. and Klawonn, F. (2002b). Improved fuzzy partitions for fuzzy regression models. *International Journal of Approximate Reasoning*. In Press.

[Höppner and Klawonn, 2002c] Höppner, F. and Klawonn, F. (2002c). Learning rules about the development of variables over time. In Leondes, C. T., editor, *Intelligent Systems: Technology and Applications*, volume IV, chapter 9, pages 201–228. CRC Press.

[Höppner et al., 2002] Höppner, F., Klawonn, F., and Eklund, P. (2002). Learning indistinguishability from data. *Soft Computing*, 6(1):6–13.

[Höppner et al., 1999] Höppner, F., Klawonn, F., Kruse, R., and Runkler, T. A. (1999). *Fuzzy Cluster Analysis*. John Wiley & Sons, Chichester, England.

[Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice-Hall.

[Joentgen et al., 1999] Joentgen, A., Mikenina, L., Weber, R., and Zimmermann, H.-J. (1999). Dynamic fuzzy data analysis based on similarity between functions. *Fuzzy Sets and Systems*, 105:81–90.

[Julian et al., 1998] Julian, P., Jordán, M., and Desages, A. (1998). Canonical piecewise-linear approximation of smooth functions. *IEEE Trans. on Circuits and Systems – I: Fundamental Theory and Applications*, 45(5):567–571.

[Kadous, 1999] Kadous, M. W. (1999). Learning comprehensible descriptions of multivariate time series. In *Proc. of the 16th Int. Conf. on Machine Learning*, pages 454–463.

[Kam and Fu, 2000] Kam, P.-S. and Fu, A. W.-C. (2000). Discovering temporal patterns for interval-based events. In *Proc. of the 2nd Int. Conf. on Data Warehousing and Knowl. Discovery*, volume 1874 of *LNCS*, pages 317–326. Springer.

[Karczewicz et al., 1995] Karczewicz, M., Hämäläinen, M., and Gabbouj, M. (1995). Multiple knot spline approximation for ECG data compression. In *Proceedings NSIP*.

[Karimi and Hamilton, 2000] Karimi, K. and Hamilton, H. J. (2000). Finding temporal relations: Causal bayesian networks vs. C4.5. In *Proc. of the 12th Int. Symp. on Methodologies for Intelligent Systems*, pages 266–273, Charlotte, NC, USA.

[Karnetzki, 1999] Karnetzki, D. (1999). *Luftdruck und Wetter*. Delius Klasing, 3rd edition.

[Keller and Klawonn, 2000] Keller, A. and Klawonn, F. (2000). Fuzzy clustering with weighting of data variables. *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 8(6):735–746.

[Keogh, 1997] Keogh, E. J. (1997). A fast and robust method for pattern matching in time series databases. In *Proceedings of 9th Int. Conf. on Tools with AI (TAI 97)*.

[Keogh et al., 2000] Keogh, E. J., Chakrabarti, K., Pazzani, M. J., and Mehrotra, S. (2000). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems Journal*, 3(3):263–286.

[Keogh and Pazzani, 1999a] Keogh, E. J. and Pazzani, M. J. (1999a). An indexing scheme for fast similarity search in large time series databases. In *Proc. of the 11th Int. Conf. on Scientific and Statistical Database Management, Cleveland, Ohio*.

[Keogh and Pazzani, 1999b] Keogh, E. J. and Pazzani, M. J. (1999b). Scaling up dynamic time warping to massive datasets. In *Proc. of the 3rd Europ. Conf. on Principles of Data Mining and Knowl. Discovery*, number 1704 in LNAI, pages 1–11, Prague, Czech Republic. Springer.

[Keogh and Pazzani, 2001] Keogh, E. J. and Pazzani, M. J. (2001). Dynamic time warping with higher order features. In *Proc. of the 1st SIAM Int. Conf. on Data Mining*, Chicago, USA.

[Keogh and Smyth, 1997] Keogh, E. J. and Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. In *Proc. of the 3rd Int. Conf. on Knowl. Discovery and Data Mining*, pages 20–24.

[Kim et al., 2000] Kim, E. D., Lam, J. M. W., and Han, J. (2000). Aim: Approximate intelligent matching for time series data. In Kambayashi, Y., Mohania, M., and Tjoa, A. M., editors, *Proc. of the 2nd Int. Conf. on Data Warehousing and Knowl. Discovery*, volume 1874 of *LNCS*, pages 347–357, London, UK. Springer.

[Klawonn and Kruse, 1995] Klawonn, F. and Kruse, R. (1995). Derivation of fuzzy classification rules from multidimensional data. In Lasker, G. and Liu, X., editors, *Advances in Intelligent Data Analysis*, pages 90–94, Windsor, Ontario.

[Klemettinen et al., 1994] Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo, A. I. (1994). Finding interesting rules from large sets of discovered association rules. In *Proc. of the 3rd Int. Conf. on Inform. and Knowl. Management*, pages 401–407.

[Konstantinov and Yoshida, 1992] Konstantinov, K. B. and Yoshida, T. (1992). Real-time qualitative analysis of the temporal shapes of (bio)process variables. *AIChE Journal: Chemical Engineering Research and Development*, 38(11):1703–1715.

[Krabs, 1969] Krabs (1969). Gleichmäßige Approximation von Funktionen. In Laugwitz, D., editor, *Überblick Mathematik*, volume 3, pages 39–69. Bibl. Inst. Mannheim.

[Kuipers, 1994] Kuipers, B. (1994). *Qualitative Reasoning – Modeling and Simulation with Incomplete Knowledge*. MIT Press.

[Li et al., 2000] Li, Y., Wang, X. S., and Jajodia, S. (2000). Discovering temporal patterns in multiple granularities. In Roddick, J. and Hornsby, K., editors, *Proc. of the 1st Int. Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining*, number 2007 in LNAI, pages 5–19, Lyon, France. Springer.

[Lin et al., 2000] Lin, W., Orgun, M. A., and Williams, G. J. (2000). Temporal data mining using multilevel-local polynomial models. In *Proc. of the 2nd Int. Conf. on Intelligent Data Engineering and Automated Learning*, volume 1983 of *LNCS*, pages 180–186. Springer.

[Lindeberg, 1993a] Lindeberg, T. (1993a). Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. *International Journal of Computer Vision*, 11(3):283–318.

[Lindeberg, 1993b] Lindeberg, T. (1993b). Effective scale: A natural unit for measuring scale-space lifetime. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(10):1068–1074.

[Lindeberg, 1994] Lindeberg, T. (1994). *Scale-Space Theory in Computer Vision*. Int. Series in Engineering and Computer Science, Robotics: Vision, Manipulation and Sensors. Kluwer Academic Publishers, Dordrecht.

[Mallat, 2001] Mallat, S. G. (2001). *A Wavelet Tour of Signal Processing*. Academic Press, Inc., 2nd edition.

[Mallat and Zhong, 1992] Mallat, S. G. and Zhong, S. (1992). Characterization of signals from multiscale edges. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(7):710–732.

[Mannila and Salmenkivi, 2001] Mannila, H. and Salmenkivi, M. (2001). Finding simple intensity descriptions from event sequence data. In *Proc. of the 7th Int. Conf. on Knowl. Discovery and Data Mining*, pages 341–346, San Francisco, USA.

[Mannila et al., 1997] Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. Technical Report 15, University of Helsinki, Finland.

[Marr and Hildreth, 1980] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proc. R. Soc. Lond.*, B 207:187–217.

[Martinelli, 1998] Martinelli, M. (1998). Pattern recognition in time-series. *Technical Analysis in Stocks & Commodities*.

[McIlraith, 1989] McIlraith, S. A. (1989). Qualitative data modeling: application of a mechanism for interpreting graphical data. *Computational Intelligence (Theory and Practice)*, 5:111–120.

[Medino-Chico et al., 2001] Medino-Chico, V., Suárez, A., and Lutsko, J. F. (2001). Backpropagation in decision trees for regression. In *Proc. of 12th Europ. Conf. on Machine Learning*, volume 2167 of *LNAI*, pages 348–359, Freiburg, Germany. Springer.

[Miller and Yang, 1997] Miller, R. J. and Yang, Y. (1997). Association rules over interval data. In *Proc. of ACM SIGMOD Int. Conf. on Data Management*, pages 452–461, Tucson, Arizona, USA.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

[Oates, 1999] Oates, T. (1999). Identifying distinctive subsequences in multivariate time series by clustering. In *Proc. of the 5th Int. Conf. on Knowl. Discovery and Data Mining*, pages 322–326.

[Pavlidis and Horowitz, 1974] Pavlidis, T. and Horowitz, S. L. (1974). Segmentation of plane curves. *IEEE Trans. on Computers*, 23(8):860–870.

[Pearl, 1984] Pearl, J. (1984). *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[Povinelli, 2000] Povinelli, R. J. (2000). Identifying temporal patterns for characterization and prediction of financial time series events. In Roddick, J. and Hornsby, K., editors, *Proc. of the 1st Int. Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining*, number 2007 in LNAI, pages 46–61.

[Pyle, 1999] Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann Publishers.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

[Rainsford and Roddick, 1999] Rainsford, C. P. and Roddick, J. F. (1999). Adding temporal semantics to association rules. In Zytkow, J. and Rauch, J., editors, *Proc. of 10th Europ. Conf. on Machine Learning*, volume 1704 of *LNAI*, pages 504–509. Springer.

[Ramsay and Silverman, 1997] Ramsay, J. O. and Silverman, B. W. (1997). *Functional Data Analysis*. Springer Series in Statistics. Springer.

[Rodriguez and Alonso, 2002] Rodriguez, J. J. and Alonso, C. J. (2002). Boosting interval-based literals: Variable length and early classification. In *Proc. of the ECAI'02 Workshop on Knowledge Discovery from (Spatio-) Temporal Data*, pages 51–62, Lyon, France.

[Rosin and West, 1995] Rosin, P. L. and West, G. A. W. (1995). Nonparametric segmentation of curves into various representations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(12):1140–1153.

[Runkler and Bezdek, 1999] Runkler, T. A. and Bezdek, J. C. (1999). Alternating cluster estimation: A new tool for clustering and function approximation. *IEEE Trans. on Fuzzy Systems*, 7(4):377–393.

[Sankoff and Kruskal, 1983] Sankoff, D. and Kruskal, J. B. (1983). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison Wesley.

[Savnik et al., 2000] Savnik, I., Lausen, G., Kahle, H.-P., Spiecker, H., and Hein, S. (2000). Algorithm for matching sets of time series. In *Int. Conf. on Principles of Data Mining and Knowledge Discovery*, pages 277–288.

[Schneider, 1990] Schneider, P. J. (1990). An algorithm for automatically fitting digitized curves. In Glassner, A. S., editor, *Graphic Gems*, pages 612–626. Academic Press, Inc.

[Sebastiani et al., 1999] Sebastiani, P., Ramoni, M., Cohen, P. R., Warwick, J., and Davis, J. (1999). Discovering dynamics using bayesian clustering. In *Proc. of the 3rd Int. Symp. on Intelligent Data Analysis*, pages 199–209, Amsterdam, The Netherlands. Springer, Berlin.

[Shahar and Musen, 1996] Shahar, Y. and Musen, M. A. (1996). Knowledge-based temporal abstraction in clinical domains. *Artificial Intelligence in Medicine*, 8:267–298.

[Shao, 1998] Shao, J. (1998). Application of an artifial neural network to improve short-term road ice forecasts. *Expert Systems With Applications*, 14:471–482.

[Shatkay, 1995] Shatkay, H. (1995). Approximate queries and representations for large data sequences. Technical Report 3, Dep. of Computer Science, Brown University.

[Shensa, 1992] Shensa, M. J. (1992). The discrete wavelet transform: Wedding the à trous and Mallat algorithms. *IEEE Trans. on Signal Processing*, 40(10):2464–2484.

[Sklansky and Gonzalez, 1980] Sklansky, J. and Gonzalez, V. (1980). Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331.

[Smyth, 1997] Smyth, P. (1997). Clustering sequences with hidden markov models. In *Advances in Neural Information Processing 9*. MIT Press.

[Smyth and Goodman, 1991] Smyth, P. and Goodman, R. M. (1991). Rule induction using information theory. In *Knowledge Discovery in Databases*, chapter 9, pages 159–176. MIT Press.

[Sprecher Energie, 1990] Sprecher Energie (1990). *MeteoLiner Kurz-Information*. Sprecher Energie, Linz. User manual for electronic barometer.

[Srikant and Agrawal, 1995] Srikant, R. and Agrawal, R. (1995). Mining generalized association rules. In *Proc. of the 21st Int. Conf. on Very Large Databases*, pages 407–419.

[Srikant and Agrawal, 1996] Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th Int. Conf. on Extending Database Technology*, pages 3–17, Avignon, France.

[Villafane et al., 1999] Villafane, R., Hua, K. A., Tran, D., and Maulik, B. (1999). Mining interval time series. In *Proc. of the 1st Int. Conf. on Data Warehousing and Knowl. Discovery*, pages 318–330. Springer.

[Villafane et al., 2000] Villafane, R., Hua, K. A., Tran, D., and Maulik, B. (2000). Knowledge discovery from series of interval events. *Journal of Intelligent Information Systems*, 15(1):71–89.

[Wang and Lee, 1998] Wang, Y.-P. and Lee, S. L. (1998). Scale-space derived from B-splines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(10):1040–1055.

[Witkin, 1983] Witkin, A. P. (1983). Scale space filtering. In *Proc. of the 8th Int. Joint Conf. on Artifial Intelligence*, pages 1019–1022, Karlsruhe, Germany.

[Xiaowei et al., 1998] Xiaowei, X., Ester, M., Kriegel, H.-P., and Sander, J. (1998). A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 324–331, Orlando, Florida.

[Yoshinari et al., 1993] Yoshinari, Y., Pedrycz, W., and Hirota, K. (1993). Construction of fuzzy models through clustering techniques. *Fuzzy Sets and Systems*, 54:157–165.

Appendix A

Appendix

This appendix contains algorithms that were referred to in the text, but would have diverted from the main thread if discussed in-place.

A.1 Continuous Piecewise Linear Approximation

Given a time series of n data points $\mathbf{x}_j = (x_j, \hat{x}_j)$ and $k+1$ knot points $\mathbf{p}_i = (p_i, \hat{p}_i)$ of a piecewise linear function $\mathbb{R} \to \mathbb{R}$. We define intervals $I_1 = [-\infty, p_2]$, $I_i = [p_i, p_{i+1}]$ for $i \in \{2, .., k-1\}$ and $I_k = [p_k, \infty]$. To denote the membership of a data point $\mathbf{x}_j$ to the i^{th} segment we use indicator variables $u_{i,j}$ with

$$u_{i,j} = \begin{cases} 1 & \text{if } x_j \in I_i \\ 0 & \text{if } x_j \notin I_i \end{cases} \tag{A.1}$$

The $u_{i,*}$ values correspond to the weight functions w_i in (2.1). Let

$$d_{i,j} = \left| \begin{pmatrix} x_j - p_i \\ \hat{x}_j - \hat{p}_i \end{pmatrix}^\top \begin{pmatrix} \hat{p}_i - \hat{p}_{i+1} \\ p_{i+1} - p_i \end{pmatrix} \right|$$

denote the scaled Euclidean distance of $\mathbf{x}_j$ to segment i. Instead of minimizing the (scaled) error function (2.4)

$$E = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{i,j} \cdot d_{i,j}^2$$

directly, we minimize the error by optimizing knot values $\hat{p}_i$ (assuming the knot points p_i to be constant) and knot points p_i (assuming $\hat{p}_i$ to be constant) alternatingly. Then both optimization steps reduce to solving a system of linear equations:

Theorem 16 *Given that the objective function J is minimized with respect to knot-values $(p_1, ..., p_{k+1})$ (resp. $(\hat{p}_1, ..., \hat{p}_{k+1})$), the equation $(p_1, ..., p_{k+1}) = -T^{-1}s$ (resp. $(\hat{p}_1, ..., \hat{p}_{k+1}) = -T^{-1}s$) with $T \in \mathbb{R}^{k+1 \times k+1}$, $s \in \mathbb{R}^{k+1}$ and*

$$\begin{aligned} s_i &= \sum_{j=1}^{n} z_j \left(u_{i,j}(\hat{l}_i - \hat{l}_{i+1})(\hat{l}_{i+1} - \hat{z}_j) + u_{i-1,j}(\hat{l}_{i-1} - \hat{l}_i)(\hat{z}_j - \hat{l}_{i-1}) \right) \\ t_{i,i-1} &= \sum_{j=1}^{n} \left(u_{i-1,j}(\hat{z}_j - \hat{l}_{i-1})(\hat{l}_i - \hat{z}_j) \right) \end{aligned}$$

$$\begin{aligned} t_{i,i} &= \sum_{j=1}^{n} \left(u_{i,j}(\hat{l}_{i+1} - \hat{z}_j)^2 + u_{i-1,j}(\hat{z}_j - \hat{l}_{i-1})^2 \right) \\ t_{i,i+1} &= \sum_{j=1}^{n} \left(u_{i,j}(\hat{z}_j - \hat{l}_i)(\hat{l}_{i+1} - \hat{z}_j) \right) \end{aligned}$$

holds ($t_{i,j} = 0$ otherwise), where $\hat{l}_i = \hat{p}_i$, $z_j = x_j$ and $\hat{z}_j = \hat{x}_j$ (resp. $\hat{l}_i = p_i$, $z_j = \hat{x}_j$, and $\hat{z}_j = x_j$).

Proof of Theorem 16: A minimum of J gives us the necessary conditions $\frac{\partial J}{\partial p_i} = 0$ and $\frac{\partial J}{\partial \hat{p}_i} = 0$. During differentiation we must take into account that the $u_{i,j}$ depend on the k_i values. Since all $u_{i,j}$ are piecewise constant, we have $\frac{\partial u_{i,j}}{\partial y} = 0$ for any y. With

$$d_{i,j} = (x_j - p_i)(\hat{p}_i - \hat{p}_{i+1}) + (\hat{x}_j - \hat{p}_i)(p_{i+1} - p_i)$$

we thus obtain

$$\frac{\partial J}{\partial y} = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{i,j} \frac{\partial d_{i,j}^2}{\partial y}$$

for $y = p_i$ and $y = \hat{p}_i$. Let us now consider the necessary condition $\frac{\partial J}{\partial p_i} = 0$:

$$\begin{aligned} \frac{\partial d_{i,j}}{\partial p_i} &= -(\hat{p}_i - \hat{p}_{i+1}) - (\hat{x}_j - \hat{p}_i) = \hat{p}_{i+1} - \hat{x}_j \\ \frac{\partial d_{i-1,j}}{\partial p_i} &= \hat{x}_j - \hat{p}_{i-1} \\ \frac{1}{2}\frac{\partial J}{\partial p_i} &= \sum_{j=1}^{n} \left(u_{i,j} d_{i,j}(\hat{p}_{i+1} - \hat{x}_j) + u_{i-1,j} d_{i-1,j}(\hat{x}_j - \hat{p}_{i-1}) \right) \\ &= \sum_{j=1}^{n} u_{i,j} \left((x_j - p_i)(\hat{p}_i - \hat{p}_{i+1}) + (\hat{x}_j - \hat{p}_i)(p_{i+1} - p_i) \right) (\hat{p}_{i+1} - \hat{x}_j) \\ &+ \sum_{j=1}^{n} u_{i-1,j} \left((x_j - p_{i-1})(\hat{p}_{i-1} - \hat{p}_i) + (\hat{x}_j - \hat{p}_{i-1})(p_i - p_{i-1}) \right) (\hat{x}_j - \hat{p}_{i-1}) \\ &= \sum_{j=1}^{n} x_j \left(u_{i,j}(\hat{p}_i - \hat{p}_{i+1})(\hat{p}_{i+1} - \hat{x}_j) + u_{i-1,j}(\hat{p}_{i-1} - \hat{p}_i)(\hat{x}_j - \hat{p}_{i-1}) \right) \\ &+ p_{i-1} \sum_{j=1}^{n} \left(u_{i-1,j}(\hat{x}_j - \hat{p}_{i-1})(\hat{p}_i - \hat{x}_j) \right) \qquad \text{(A.2)} \\ &+ p_i \sum_{j=1}^{n} \left(u_{i,j}(\hat{p}_{i+1} - \hat{x}_j)^2 + u_{i-1,j}(\hat{x}_j - \hat{p}_{i-1})^2 \right) \\ &+ p_{i+1} \sum_{j=1}^{n} \left(u_{i,j}(\hat{x}_j - \hat{p}_i)(\hat{p}_{i+1} - \hat{x}_j) \right) \qquad \text{(A.3)} \end{aligned}$$

Thus, $\frac{\partial J}{\partial p_i} = 0$ for $i \in \{1, .., k+1\}$ defines a system of linear equations $Tq + s = 0$ with tridiagonal matrix

$$T = \begin{pmatrix} t_{1,1} & t_{1,2} & & & & & & \\ t_{2,1} & t_{2,2} & t_{2,3} & & & & & \\ & & \ddots & & & & & \\ & & t_{i,i-1} & t_{i,i} & t_{i,i+1} & & & \\ & & & & \ddots & & & \\ & & & & t_{k,k-1} & t_{k,k} & t_{k,k+1} \\ & & & & & t_{k+1,k} & t_{k+1,k+1} \end{pmatrix}$$

and $q = (p_1, ..., p_{k+1})$, $t_{i,j}$ and s_i as given in the theorem.

We do not have to differentiate J with respect to $\hat{p}_i$ explicitly, thanks to the following observation: If we exchange all x/p with $\hat{x}/\hat{p}$ and vice versa in $d_{i,j}$ we obtain $-d_{i,j}$. Since $d_{i,j}$ occurs only squared in J, the derivative of $\sum_{j=1}^{n} \sum_{i=1}^{k} u_{i,j}(-d_{i,j})^2$ with respect to $\hat{p}_i$ is identical to the derivative of J with respect to p_i. Thus, we obtain $\frac{\partial J}{\partial \hat{p}_i}$ from $\frac{\partial J}{\partial p_i}$ by simply replacing all x/p with $\hat{x}/\hat{p}$ and vice versa. ■

When solving for knot locations p_i it seems reasonable to require $p_1 = \min_j x_j$ and $p_{k+1} = \max_j x_j$. In this case, the system of linear equations consists of the equations $\frac{\partial J}{\partial p_i}$ for $2 \leq i \leq k$ (dimension $k-1$). Now, for $i = 2$ (resp. $i = k$) the term (A.2) (resp. (A.3)) is constant and has to be respected by s_2 (resp. s_k). The complete algorithm as discussed in section 2.2.1.2 is given in figure A.1.

---

1 initialize knot points $\mathbf{p}_i$
2 select termination threshold ε and maximum no. of iterations i_{max}
3 **repeat**
4 update knot values $\hat{p}_i$ according to Theorem 16
5 update knot location p_i according to Theorem 16
6 **until** change in p_i and $\hat{p}_i$ drops below ε or i_{max} has been reached

---

Figure A.1: Piecewise linear function approximation algorithm.

A.2 Mixture of Regression Models

In this section we briefly discuss how regression (e.g. [Bezdek, 1980]) and clustering (e.g. [Höppner et al., 1999]) can be combined to get an adaptive approximation as discussed in section 2.2.1.3 and [Höppner and Klawonn, 2000b, Höppner and Klawonn, 2002b]. Clustering aims at grouping similar objects into the same cluster and dissimilar objects into different clusters. The partitioning of data objects is modelled via a membership degree of data object $x_j \in X$, $j \in \{1, ..., n\}$, to cluster $p_i \in P$, $i \in \{1, ..., c\}$, denoted by $u_{i,j} \in [0, 1]$. Quite often data objects cannot be assigned unambigously to a cluster, which is the reason for considering "fuzzy" memberships between 0 and 1. Given a distance $d(x_j, p_i)$ of a data object x_j to a cluster prototype p_i, the membership degrees are obtained by minimizing the objective function

$$J_m(P, U;\ X) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{i,j}^m d^2(x_j, p_i) \tag{A.4}$$

where the so-called "fuzzifier" m is chosen in advance and influences the fuzziness of the final partition (crisp as $m \to 1$ and totally fuzzy as $m \to \infty$; common values for m are within 1.5 and 4, 2 is most frequently used). J_m becomes minimal if high membership degrees $u_{i,j} \approx 1$ are assigned to data objects close to the cluster prototype ($d(x_j, p_i) \approx 0$) and low membership degrees $u_{i,j} \approx 0$ are assigned to data objects far away from the prototype ($d(x_j, p_i) \gg 0$). The objective function is minimized iteratively subject to the constraints

$$\forall_{1 \le j \le n} : \sum_{i=1}^{c} u_{i,j} = 1, \quad \forall_{1 \le i \le c} : \sum_{j=1}^{n} u_{i,j} > 0 \tag{A.5}$$

which guarantee that every data object is considered with the same overall weight of 1, and that none of the clusters is empty.

Minimization with respect to $u_{i,j}$ and p_i is done separately. The necessary conditions for a minimum yield update equations for both half-steps. Independent of the choice of the distance function and the prototypes, the membership update equation is

$$u_{i,j} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d^2(x_j, p_i)}{d^2(x_j, p_k)} \right)^{\frac{1}{m-1}}} \tag{A.6}$$

In the most simple case of the fuzzy c-means clustering algorithm (FCM), where the prototypes – to be interpreted as cluster centres – are vectors of the same dimension as the data vectors and the distance function is the Euclidean distance d_E, we obtain

$$p_i = \frac{\sum_{j=1}^{n} u_{i,j}^m x_j}{\sum_{j=1}^{n} u_{i,j}^m}. \tag{A.7}$$

Similarly a fuzzy c-regression models algorithm (FCRM) can be defined [Hathaway and Bezdek, 1993], which uses polynomials as cluster prototypes. With real functions $\mathbb{R} \to \mathbb{R}$ the cluster models are characterized by the coefficients of the polynomial, that is, the prototypes are elements of $\mathbb{R}^{q+1}$ where q is the degree of the polynomials. The Euclidean distance d_E of FCM is replaced by the residual error $|y - h(x)|$ of a data object (x, y) (consisting of input value x and output value y) to the polynomial h. For simplicity, we consider extended data objects $\hat{x}$ which have an additional component $\hat{x}_0 \equiv 1$. Then, the distance function can be written as

$$d^2((x_j, y_j), p_i) \;=\; \left(y_j - p_i^\top \hat{x}_j\right)^2.$$

For multiple inputs $\hat{x}_j$ has to be extended further, for instance for $x_j = (a, b)$ we have $\hat{x}_j = (1, a, b, ab, a^2, b^2)$ such that all coefficients of the polynomial can be represented by an element of p_i. The coefficients p_i are obtained in the same fashion as the cluster centres of FCM before, we only have to replace the prototype update equation according to the modified distance function:

$$p_i = \left(\sum_{j=1}^{n} u_{i,j}^m (\hat{x}_j \hat{x}_j^\top) \right)^{-1} \left(\sum_{j=1}^{n} u_{i,j}^m y_j \hat{x}_j \right) \tag{A.8}$$

With FCRM clusters the neighbourhood of the output value y_i to a polynomial $f_i(x_i)$ alone decides about the membership to the clusters, the input values x_i of the data objects in the cluster are not necessarily neighboured as we would expect from a time series segmentation. But we can achieve this by simultaneously clustering the input values using FCM. Since both algorithms (clustering via FCM and regression

via FCRM) are objective function-based, their combination is straightforward. The combined algorithm uses the sum of both distance functions:

$$d^2(\,(x_j, y_j)\,,\,(p_i, q_i)\,)\;=\;\underbrace{\|x_j - p_i\|^2}_{\text{FCM distance}}\;+\;\underbrace{\left(y_j - q_i^\top \hat{x}_j\right)^2}_{\text{FCRM distance}}\,. \tag{A.9}$$

The FCM distances are taken with respect to the input value x_j and cluster centre p_i, while the FCRM distances are taken with respect to the given output value y_j and the value of the polynomial at $\hat{x_j}$ with coefficients q_i. The algorithm is sketched in Fig. A.2.

---

1 choose number of clusters c;
2 choose termination threshold $\varepsilon > 0$;
3 choose $\eta > 0$;
4 initialize prototypes p_i, q_i;
5 **repeat**
6 update memberships using (A.6) and distances (A.9);
7 update prototypes p_i using (A.7);
8 update prototypes q_i using (A.8);
9 **until** change in memberships drops below ε;

Figure A.2: The FM algorithm.

Since there are no dependencies between the parameters of the modified clustering and regression prototypes (p_i and q_i), the same prototype update equations hold for the combined algorithm. Nevertheless, cluster centres and polynomials influence each other indirectly by means of the membership degrees, which depend on the distance to both models. (A different way to combine FCM and linear FCRM can be found in [Chen et al., 1998].)

A.3 Variance Estimation for Weighted Least-Squares

Whenever least-squares fitting is performed (cf. methods in chapter 2), we can weight the residuals by the variance. Instead of minimizing

$$\sum_{i=1}^{n}(x_i - f(x))^2,\ \text{we use}\ \sum_{i=1}^{n}\left(\frac{x_i - f(x)}{\sigma_i}\right)^2\ \text{or}\ \sum_{i=1}^{n} w_i \cdot (x_i - f(x))^2$$

with $w_i = \frac{1}{\sigma_i^2}$. If we have additional information about the local variance in the data we can avoid overfitting, because the larger error (due to noise) will be downscaled by the smaller weights w_i. Traditionally, σ_i denotes the variance of a measurement x_i, that is, the variance of x_i when measured multiple times. However, we have only a single time series available and therefore cannot calculate this σ_i.

Under the additional assumptions that (a) the variance changes slowly over time and (b) that the function is smooth, we can do a local estimation of σ_i. We assume smoothness in the sense that we can approximate the time series locally (e.g. within a neighbourhood of n time steps) by a linear segment. Then, we use the variance of the errors as σ_i^2. In the vicinity of local extrema the goodness of fit of a linear function will degrade regardless of the amount of noise. We therefore perform a

left-side, center, and right-side fitting, that is, we use the neighbourhoods

$$U_{left}(x_i) = \{x_{i-2n}, x_{i-2n+1}, ..., x_i\}$$
$$U_{center}(x_i) = \{x_{i-n}, ..., x_i, ..., x_{i+n}\}$$
$$U_{right}(x_i) = \{x_i, x_{i+1}, ..., x_{i-2n}\}$$

to fit three regression lines. The minimum variance is used as an estimate of σ_i^2. This approach nicely supports variance estimation in the presence of discontinuities, since there is always one neighbourhood that does not contain the discontinuity.

A.4 Precalculation of the Observation Interval

Given a normalized pattern P, we can identify two bounds of the participating intervals from which we obtain the observation interval, as it has been discussed in section 5.2, page 98. In this section we give an algorithm that determines these two bounds. The algorithm appears comparatively complicated due to the fact that we have to find these bounds simply from the qualitative relationships of the intervals. For each interval, it determines hypothetical left and right bounds, such that all interval relationships hold. From these interval bounds the bound next to the leftmost and next to the rightmost can be determined, as it has been done before in figure 3.9. The left and right bounds may also be useful for a visualization of temporal patterns.

```
1 proc find_vis_bounds(→ P, ← left[], ← right[], ← start, ← end)
2    init_left_bounds(P, left);
3    determine_right_bounds(P, left, right);
4    find_bounds(P, left, right, start, end);
5 .
```

Figure A.3: Main program.

The main algorithm in figure A.3 takes the normalized pattern and returns to integer arrays $left$ and $right$ of size $\dim(P)$. A pictorial representation of P can be obtained from drawing interval #i from $left[i]$ to $right[i]$. The return value *start* contains both, the interval number i of the bound next to the leftmost bound as well as the side s of the bound (left bound is side 0, right bound is side 1). The return value *start* is then $2i + s$. Similar for *end*.

```
1 proc init_left_bounds(→ P, ← left[])
2    x ↤ 0; left[0] ↤ x;
3    for i ↤ 2 to dim(P) do
4       if P.R[i − 1, i] ≠ equals ∧ P.R[i − 1, i] ≠ starts then x ↤ x + 1 fi;
5       left[i] ↤ x;
6    od
7 .
```

Figure A.4: Determination of left bounds.

The first and simpler step is to initialize the left bounds. There are only two cases in a normalized pattern P where the left bounds of consecutive intervals coincide:

only for a *starts* or *equals* relationship. In all other cases, there is a gap between the two left bounds of neighbouring intervals. The $left[]$ array is initialized in this way as shown in figure A.4 (where a unit gap length of 1 is used).

```
 1 proc determine_right_bounds(→ P, ↔ left[], ← right[])
 2   right[dim(P)] ↤ left[dim(P)] + 1;
 3   if dim(P) = 1 then return fi;
 4   maxx ↤ right[dim(P)];
 5   for i ↤ dim(P) − 1 to 1 do
 6       j ↤ dim(P); l ↤ left[i]; r ↤ maxx + 1;
 7       while (l < r) ∧ (i < j) do
 8             if P.R[i, j] = before then r ↤ min{r, left[j]};
 9             elsif P.R[i, j] = starts ∨ P.R[i, j] = overlaps
10                   then r ↤ min{r, right[j]}; l ↤ max{l, left[j]};
11             elsif P.R[i, j] = meets then r ↤ left[j]; l ↤ left[j];
12             elsif P.R[i, j] = equals ∨ P.R[i, j] = is-finished-by
13                   then r ↤ right[j]; l ↤ right[j];
14             elsif P.R[i, j] = contains then l ↤ max{l, right[j]}; fi
15             j ↤ j − 1;
16       od
17       if l = r
18         then right[i] ↤ l;
19          else for k ↤ 1 to dim(P) do
20                   if left[k] ≥ r then ++left[k] fi
21                   if k ≥ i + 1 ∧ right[k] ≥ r then ++right[k] fi
22               od
23               right[i] ↤ r; maxx ↤ maxx + 1;
24       fi
25   od
26 .
```

Figure A.5: Determination of right (and correcting of left) bounds.

The right bounds are somewhat more complicated, because the right bound of an interval may coincide with any (left or right) bound of any interval with higher index. In the algorithm in figure A.5 we start from the highest index ($right[\dim(P)]$) and initialize it to $1+left[\dim(P)]$ (intervals must have non-zero length). If the pattern size is 1, we already done. In $maxx$ we remember the maximum bound we have so far (which is at this stage $right[\dim(P)]$, but not necessarily in later stages, because other intervals than the last one may contain the last interval, for example).

Then, we consider the right bound x for all other intervals i (from right $i = \dim(P)-1$ to left $i = 1$). For each i, we examine the intervals $j \in [i, \dim(P)]$ to find left and right bounds l and r that locate the right bound of the current interval i. If we end up with values $l \neq r$ we have $l < x < r$ (we have to shift some bounds to the right in order to insert the new bound x, line 19ff), if $l = r$ we have $l = x = r$ (and no shifting is necessary, line 18). Any relationship between interval i and j may help in further restricting l and r, for example, if $P.R[i, j] = before$ then we know that the right bound of interval i is smaller than the left bound of interval j and thus $r \leftarrow \min\{r, left[j]\}$ (line 8). The other cases are handled similarly.

Finally, we can use the integer bounds we have determined for $left[]$ and $right[]$ to find the bound next to the minium and maximum bound, as required in section 3.2.1 (cf. algorithm in figure 3.9). This is done in the last procedure shown in

figure A.6. The case that the first or last interval bound may occur more than once requires some additional treatment. Up to line 12 the behaviour is essentially that of the algorithm in figure 3.9. At the end of the loop, *nearstartpos* and *nearendpos* are the (integer) bounds next to the leftmost and next to the rightmost bound. In the last few lines we iterate once more over the intervals to determine the index of the interval and the side that uses the identified positions *nearstartpos* and *nearendpos*.

```
1  proc find_bounds(→ P, → left[], → right[], ← start, ← end)
2    nearstartpos ↤ 1;
3    if dim(P) > 1 ∧ left[1] ↤ 0 then nearstartpos ↤ 0; fi
4    nearendpos ↤ left[dim(P)]; endpos ↤ right[dim(P)];
5    for k ↤ 1 to dim(P) do
6        x ↤ right[k];
7        if x > endpos then nearendpos ↤ endpos; endpos ↤ x;
8        elsif x > nearendpos then nearendpos ↤ x; fi
9        x ↤ left[k];
10       if x > endpos then nearendpos ↤ endpos; endpos ↤ x;
11       elsif x > nearendpos then nearendpos ↤ x; fi
12   od
13   start ↤ −1; end ↤ −1;
14   for k ↤ 1 to dim(P) do
15       if end < 0
16         then if left[k] = nearstartpos then end ↤ 2k + 0;
17                 elsif right[k] = nearstartpos then end ↤ 2k + 1; fi
18       fi
19       if start < 0
20         then if left[k] = nearendpos then start ↤ 2k + 0;
21                 elsif right[k] = nearendpos then start ↤ 2k + 1; fi
22       fi
23   od
24 .
```

Figure A.6: Determination of interval number and side.

Index